

《linux应用实践》

综合实践

题 目：命名管道实现多客户端服务端

基于TCP的聊天程序

Tensorflow实现聊天机器人

学 院： 计算机与信息安全学院

专 业： 计算机科学与技术

姓 名： 王 涛

学 号： 1600300128

指导教师： 张瑞霞

2019年1月8日

成绩评定标准及成绩

1. 能按照格式进行写作，无抄袭现象（10分）
2. 报告内容行文通畅，有条理性，无错别字，结构严谨。（10分）
3. 能够按照格式要求、排版要求和字数要求等，有需求分析，系统设计与实现，系统测试和参考文献。（10分）
4. 在验收过程中，能合理的回答问题（20分）
5. 软件能正常运行，实现所提出的功能（40分）
6. 软件代码规范性较好（5分）
7. 具有自己的创新或特色（5分）

总成绩：

**摘要**

初级题目采用命名管道编程技术，实现了简单的客户端、服务端模式，其主要功能包括多个客户端向服务端发起请求。

中级题目采用网络编程的socket技术，实现了基于TCP的多线程聊天程序，其主要功能包括：多个客户端同时在线聊天，服务端群发消息，服务端部署在服务器上，可以实现不同主机间聊天。

高级题目采用深度学习框架Tensorflow及Seq2Seq模型，实现了英文版的智能聊天机器人，其主要功能是可以与人智能对话，一问一答式自动回复聊天内容。

关键字：命名管道；聊天室；多线程；Tensorflow；聊天机器人；

**目录**

[1. 绪论 4](#_Toc534740212)

[1.1. 需求分析 4](#_Toc534740213)

[1.1.1 命名管道实现多客户端请求 4](#_Toc534740214)

[1.1.2 网络聊天室 4](#_Toc534740215)

[1.1.3 Tensorflow智能聊天机器人 4](#_Toc534740216)

[1.2. 开发环境 4](#_Toc534740217)

[1.3. 相关理论 5](#_Toc534740218)

[1.3.1 命名管道 5](#_Toc534740219)

[1.3.2 scoket网络编程原理 5](#_Toc534740220)

[1.3.3 Tensorflow基本原理 6](#_Toc534740221)

[1.3.4 Seq2Seq模型原理 6](#_Toc534740222)

[2. 设计与实现 7](#_Toc534740223)

[2.1. 概要设计 7](#_Toc534740224)

[2.1.1 命名管道多客户端请求设计 7](#_Toc534740225)

[2.1.2 Tensorflow模型训练与网络聊天 7](#_Toc534740226)

[2.2. 详细设计与实现 8](#_Toc534740227)

[2.2.1 命名管道多客户端服务端实现 8](#_Toc534740228)

[2.2.2 网络聊天室的设计与实现 10](#_Toc534740229)

[2.2.3 Tensorflow聊天机器人的实现 17](#_Toc534740230)

[3. 系统测试 23](#_Toc534740231)

[3.1. 命名管道服务端客户端测试 23](#_Toc534740232)

[3.2. 多人聊天测试 23](#_Toc534740233)

[3.3. AI聊天测试 25](#_Toc534740234)

[4. 总结 26](#_Toc534740235)

[5. 参考文献 27](#_Toc534740236)

# 绪论

随着计算机网络日新月异的发展，人们的交流方式已经发生了天翻地覆的变化。传统的交流方式，如：信件、电报、电话等已经难以满足人们的交流要求，互联网上即时的和好友取得联系，已经成为当今社会人们主流的联系方式。因此，两台计算机之间进行即时通讯、发送文件等交流方式已经成为必然潮流。在最近几年，随着深度学习等相关技术的成熟，人工智能发展愈发火热，深度学习中一个很有名的框架就是Tensorflow，它的功能非常强大，可以被应用于自然语言处理、图像识别、语音识别等诸多领域。它可在小到一部智能手机、大到数千台数据中心服务器的各种设备上运行。使用Tensorflow开发的应用也数不胜数，给人们生活带来了很大的价值。

此次，将使用Java的Socket网络编程技术利用多线程开发以一个多人聊天室，同时使用Tensorflow从环境搭建到实际应用开发，利用Seq2Seq模型训练一个智能问答机器人将其加入到多人聊天之中，实现多人在线聊天与机器人智能对话等功能。

## 需求分析

### 1.1.1 命名管道实现多客户端请求

命名管道是以管道文件的形式实现数据读写，命名管道是原子性操作，多个客户端向服务端发送请求可以通过写入命名管道，服务端依次接受请求。

### 1.1.2 网络聊天室

实现网络聊天的功能，采用Java Socket网络编程技术，服务器与客户端采用了TCP/IP连接方式，在具体的设计聊天方案时可将所有信息发往服务器端，再由服务器进行处理，服务器端是所有信息的中心。这期间需要用到多线程技术，针对每个客户端需开启一个线程、服务端和每个客户端也都要开启接收消息的进程。

### 1.1.3 Tensorflow智能聊天机器人

首先是Tensorflow环境的安装部署，tensorflow依赖于python环境，安装python可使用Anaconda安装，它包含了python和其他很丰富的数据科学计算包，同时可以管理不同环境。Python环境安装部署好，使用pip安装tensorflow。之后是使用tensorflow训练聊天机器人可分为如下几步：收集数据、数据预处理、模型构建、模型训练、模型测试等阶段。tensorflow训练模型使用的是python语言，而网络聊天室使用Java语言开发，不能直接使用模型。可使用python单独开启智能聊天的服务供客户端请求访问。

## 开发环境

操作系统：Ubuntu16.04

相关环境：Anaconda3 5.2.0（Python3.6.x）

JDK1.8

Tensorflow1.4

开发工具：Intellij IDEA、Jetbrains PyCharm

## 相关理论

### 1.3.1 命名管道

命名管道又名命名管线（Named Pipes），是一种简单的进程间通信（IPC）机制，Microsoft Windows大都提供了对它的支持（但不包括Windows CE）。命名管道可在同一台计算机的不同进程之间或在跨越一个网络的不同计算机的不同进程之间，支持可靠的、单向或双向的数据通信。

### 1.3.2 scoket网络编程原理

套接口有三种类型：流式套接口、数据报套接口及原始套接口。流式套接口定义了一种可靠的面向连接的服务,实现了无差错无重复的顺序数据传输.数据报套接口定义了一种无连接的服务,数据通过相互独立的报文进行传输,是无序的,并且不保证可靠,无差错.原始套接口允许对低层协议如IP或ICMP直接访问,主要用于新的网络协议实现的测试等。面向连接服务器处理的请求往往比较复杂，不是一来一去的请求应答所能解决的，而且往往是并发服务器。使用面向连接的套接口编程, 其时序可以通过下图来表示：

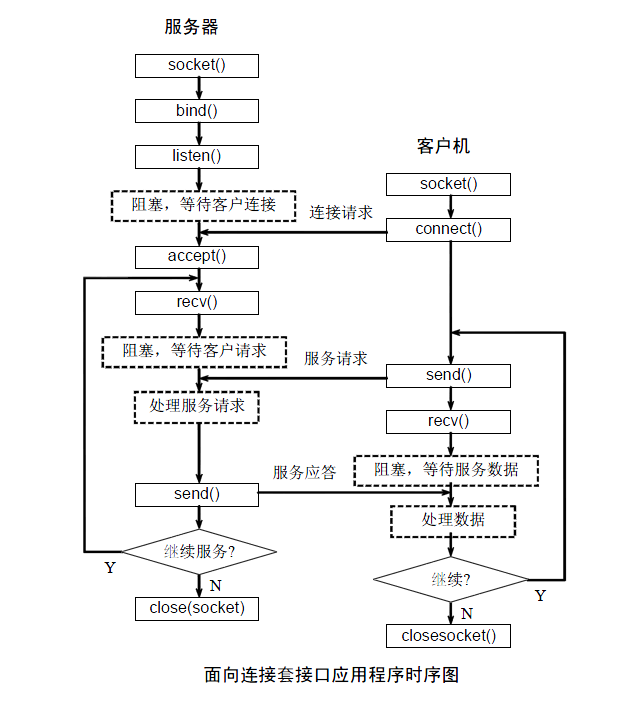


图1-1 Socket应用流程图

套接口工作过程如下：服务器首先启动,通过调用socket()建立一个套接口,然后调用bind()将该套接口和本地网络地址联系在一起,再调用listen()使套接口做好侦听的准备,并规定它的请求队列的长度,之后就调用accept()来接收连接.客户在建立套接口后就可调用connect()和服务器建立连接.连接一旦建立,客户机和服务器之间就可以通过调用send()和recv()来发送和接收数据.最后,待数据传送结束后,双方调用close()关闭套接口。

### 1.3.3 Tensorflow基本原理

TensorFlow 是一个采用数据流图，用于数值计算的开源软件库，数据流图用结点和线的有向图来描述数学计算。节点一般用来表示施加的数学操作，但也可以表示数据输入的起点/输出的终点，或者是读取/写入持久变量的终点。线表示节点之间的输入/输出关系。这些数据线可以输运size可动态调整的多维数据数组，即张量。张量从图中流过的直观图像是这个工具取名为Tensorflow的原因。一旦输入端的所有张量准备好，节点将被分配到各种计算设备完成异步并行地执行运算。

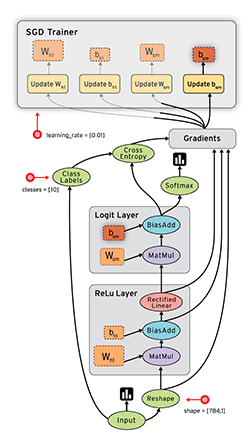


图1-2 Tensorflow原理图解

### 1.3.4 Seq2Seq模型原理

Seq2Seq解决问题的主要思路是通过深度神经网络模型将一个作为输入的序列映射为一个作为输出的序列，这一过程由编码输入（encoder）与解码输出(decoder)两个环节组成。最基础的Seq2Seq模型包含了三个部分，即Encoder、Decoder以及连接两者的中间状态向量C，Encoder通过学习输入，将其编码成一个固定大小的状态向量c，继而将c传给Decoder，Decoder再通过对状态向量c的学习来进行输出。大致如下：

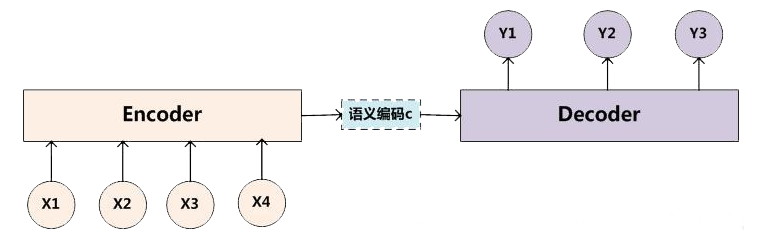


图1-3 Seq2seq原理示意图

# 设计与实现

## 概要设计

### 2.1.1 命名管道多客户端请求设计

服务端和客户端在开启时可将命名管道名称传入，客户端向命名管道写入请求数据、服务端从命名管道读取数据。便可实现多客户端的请求。如下图所示。

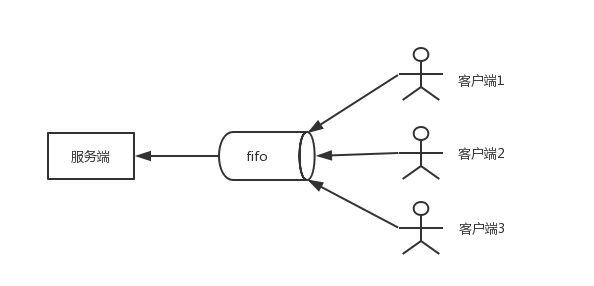


图2-1 命名管道概要设计

### 2.1.2 Tensorflow模型训练与网络聊天

使用Tensorflow和seq2seq训练聊天机器人模型，主要分为以下步骤，根据具体步骤写出相应代码，最终得到模型。得到模型以后需要将其整合到聊天室之中。详细的流程如下图所示。

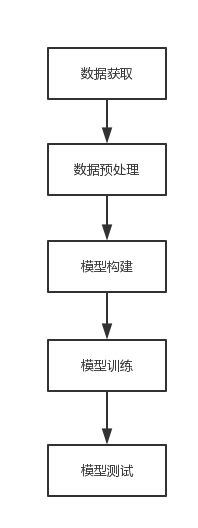


图2-2 机器人训练步骤

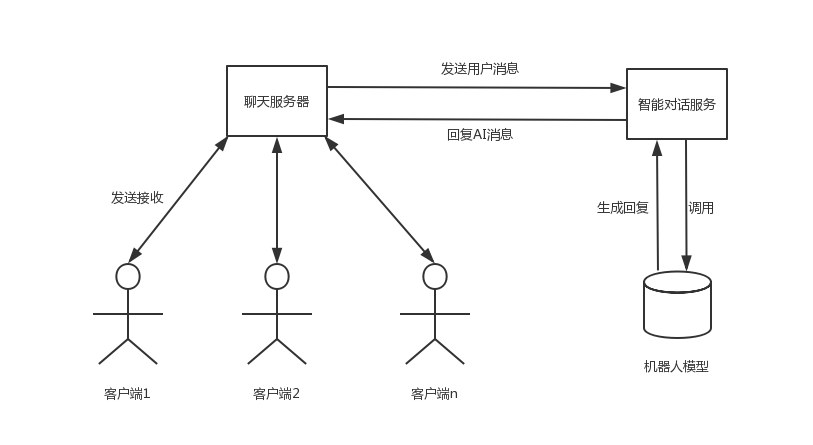


图2-3 系统流程图

## 详细设计与实现

### 2.2.1 命名管道多客户端服务端实现

客户端server.c

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#define MAXSIZE 100

struct MSG

{

char id[20];

char str[MAXSIZE];

};

void main(int argc,char \*argv[])

{

struct MSG msg;

int pfd;

char comfifo[MAXSIZE];

if(argc < 2)

{

printf("argument error!\n");

exit(1);

}

if((access(argv[1],F\_OK)) == -1)

{

if(mkfifo(argv[1],0666) < 0)

{

perror("fail to comman fifo");

exit(1);

}

}

if( (pfd = open(argv[1],O\_RDONLY)) < 0) //打开公共管道

{

perror("fail to open comman fifo");

exit(1);

}

while(1)

{

read(pfd,comfifo,sizeof(comfifo)); //从公共管道读内容

char \*s = strtok(comfifo," ");

strcpy(msg.id,s);

s = strtok(NULL," ");

strcpy(msg.str,s);

printf("recv from client: %s msg:%s\n",msg.id,msg.str);

}

exit(0);

}

客户端client.c

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>

#define MAXSIZE 100

int main(int argc,char \*argv[])

{

int pfd;

char comfifo[MAXSIZE];

char msg[MAXSIZE];

if(argc < 2)

{

printf("argument error!");

exit(1);

}

if( (pfd = open(argv[1],O\_WRONLY)) < 0) //服务端公共管道

{

perror("fail to open comman fifo file");

exit(1);

}

while(1)

{

printf("输入发送内容：");

fgets(msg,sizeof(msg),stdin); //输入发送内容

sprintf(comfifo,"%d %s",getpid(),msg);

printf("send to server :%s\n",comfifo);

write(pfd,comfifo,strlen(comfifo) + 1); //写入公共管道

}

exit(0);

}

### 2.2.2 网络聊天室的设计与实现

网络聊天室的设计主要难点在于多线程的掌控，要实现多人聊天，服务端必须引进多线程服务。对于服务端，需要定义如下几类线程：

（1）主服务线程：主要用于建立服务端套接字，初始化基本信息，与AI服务器建立连接等。同时要不断监听客户端连接，动态控制客户端列表。主要代码如下：

// 服务主线程

class ServerThread extends Thread {

private ServerSocket serverSocket;

private int max;// 人数上限

// 服务器线程的构造方法

public ServerThread(ServerSocket serverSocket, int max) {

this.serverSocket = serverSocket;

this.max = max;

}

public void run() {

while (true) {// 不停的等待客户端的链接

try {

Socket socket = serverSocket.accept();

if (clients.size() == max) {// 如果已达人数上限

BufferedReader r = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter w = new PrintWriter(socket.getOutputStream());

// 接收客户端的基本用户信息

String inf = r.readLine();

StringTokenizer st = new StringTokenizer(inf, "@");

User user = new User(st.nextToken(), st.nextToken());

// 反馈连接成功信息

w.println("MAX@服务器：对不起，" + user.getName() + user.getIp() + "，服务器在线人数已达上限，请稍后尝试连接！");

w.flush();

// 释放资源

r.close();

w.close();

socket.close();

continue;

}

ClientThread client = new ClientThread(socket);

client.start();// 开启对此客户端服务的线程

clients.add(client);

userList.add(client.getUser());// 更新在线列表

System.out.println(client.getUser().getName() + client.getUser().getIp() + "上线!\r\n");

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

（2）客户服务线程：主要用于不断接收各个客户端发送的消息并转发给其他用户，反馈给客户端连接信息，完成用户登陆退出等通知信息。关键代码如下：

// 为一个客户端服务的线程

class ClientThread extends Thread {

private Socket socket;

private BufferedReader reader;

private PrintWriter writer;

private User user;

public BufferedReader getReader() {

return reader;

}

public PrintWriter getWriter() {

return writer;

}

public User getUser() {

return user;

}

// 客户端线程的构造方法

public ClientThread(Socket socket) {

try {

this.socket = socket;

reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

writer = new PrintWriter(socket.getOutputStream());

// 接收客户端的基本用户信息

String inf = reader.readLine();

StringTokenizer st = new StringTokenizer(inf, "@");

user = new User(st.nextToken(), st.nextToken());

// 反馈连接成功信息

writer.println(user.getName() + user.getIp() + "与服务器连接成功!");

writer.flush();

// 反馈当前在线用户信息

if (clients.size() > 0) {

String temp = "";

for (int i = clients.size() - 1; i >= 0; i--) {

temp += (clients.get(i).getUser().getName() + "/" + clients.get(i).getUser().getIp()) + "@";

}

writer.println("USERLIST@" + clients.size() + "@" + temp);

writer.flush();

}

// 向所有在线用户发送该用户上线命令

for (int i = clients.size() - 1; i >= 0; i--) {

clients.get(i).getWriter().println("ADD@" + user.getName() + user.getIp());

clients.get(i).getWriter().flush();

}

} catch (IOException e) {

e.printStackTrace();

}

}

@SuppressWarnings("deprecation")

public void run() {// 不断接收客户端的消息，进行处理。

String message = null;

String res = null;

while (true) {

try {

message = reader.readLine();// 接收客户端消息

if(robot){

StringTokenizer stringTokenizer = new StringTokenizer(message, "@");

String source = stringTokenizer.nextToken();

String owner = stringTokenizer.nextToken();

String content = stringTokenizer.nextToken();

chatWriter.println(content+"over"); //将客户端消息发送给机器人

}

if (message.equals("CLOSE"))// 下线命令

{

System.out.println(this.getUser().getName() + this.getUser().getIp() + "下线!\r\n");

// 断开连接释放资源

reader.close();

writer.close();

socket.close();

// 向所有在线用户发送该用户的下线命令

for (int i = clients.size() - 1; i >= 0; i--) {

clients.get(i).getWriter().println("DELETE@" + user.getName());

clients.get(i).getWriter().flush();

}

userList.remove(this.getUser());

// 删除此条客户端服务线程

for (int i = clients.size() - 1; i >= 0; i--) {

if (clients.get(i).getUser() == user) {

ClientThread temp = clients.get(i);

clients.remove(i);// 删除此用户的服务线程

temp.stop();// 停止这条服务线程

return;

}

}

} else {

dispatcherMessage(message);// 转发消息

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

（3）AI消息线程：主要用于接收AI服务器的返回消息。

//接收AI服务器消息的线程

class ChatMessageThread extends Thread{

private BufferedReader reader;

public ChatMessageThread(BufferedReader reader){

this.reader = reader;

}

public void run() {

String message = "";

while (true) {

try {

message = reader.readLine();

dispatcherMessage(message);

} catch (Exception e) {

e.printStackTrace();

}

}

}

}

（4）服务端操作线程：主要功能包括群发消息、打开AI聊天、查看用户列表、关闭服务等操作。

//操作线程

class OptionThread extends Thread{

@Override

public void run() {

String msg = "";

Scanner sc = new Scanner(System.in);

while (isStart){

System.out.println("选择操作：1:当前用户列表\t2:群发消息\t3:聊天机器人\t0:关闭服务");

int op = sc.nextInt();

switch (op){

case 0:

closeServer();break;

case 1:

showUserList();break;

case 2:

System.out.print("输入群发消息：");

msg = sc.next();

send(msg);break;

case 3:

System.out.print("选择是否开启[y/n]");

msg = sc.next();

if(msg.equals("y")||msg.equals("true")){

System.out.println("聊天机器人已开启");

controlRobot(true);

}else{

System.out.println("聊天机器人已关闭");

controlRobot(false);

}

default:break;

}

}

}

}

客户端的线程主要是一个不断接收客户端消息的线程。

// 不断接收消息的线程

class MessageThread extends Thread {

private BufferedReader reader;

private JTextArea msgList;

// 接收消息线程的构造方法

public MessageThread(BufferedReader reader, JTextArea msgList) {

this.reader = reader;

this.msgList = msgList;

}

// 被动的关闭连接

public synchronized void closeCon() throws Exception {

// 清空用户列表

listModel.removeAllElements();

// 被动的关闭连接释放资源

if (reader != null) {

reader.close();

}

if (writer != null) {

writer.close();

}

if (socket != null) {

socket.close();

}

isConnected = false;// 修改状态为断开

}

public void run() {

String message = "";

while (true) {

try {

message = reader.readLine();

StringTokenizer stringTokenizer = new StringTokenizer(message, "/@");

String command = stringTokenizer.nextToken();// 命令

if (command.equals("CLOSE"))// 服务器已关闭命令

{

msgList.append("服务器已关闭!\r\n");

closeCon();// 被动的关闭连接

return;// 结束线程

} else if (command.equals("ADD")) {// 有用户上线更新在线列表

String username = "";

String userIp = "";

if ((username = stringTokenizer.nextToken()) != null && (userIp = stringTokenizer.nextToken()) != null) {

User user = new User(username, userIp);

onLineUsers.put(username, user);

listModel.addElement(username);

}

} else if (command.equals("DELETE")) {// 有用户下线更新在线列表

String username = stringTokenizer.nextToken();

User user = (User) onLineUsers.get(username);

onLineUsers.remove(user);

listModel.removeElement(username);

} else if (command.equals("USERLIST")) {// 加载在线用户列表

int size = Integer.parseInt(stringTokenizer.nextToken());

String username = null;

String userIp = null;

for (int i = 0; i < size; i++) {

username = stringTokenizer.nextToken();

userIp = stringTokenizer.nextToken();

User user = new User(username, userIp);

onLineUsers.put(username, user);

listModel.addElement(username);

}

} else if (command.equals("MAX")) {// 人数已达上限

msgList.append(stringTokenizer.nextToken() + stringTokenizer.nextToken() + "\r\n");

closeCon();// 被动的关闭连接

JOptionPane.showMessageDialog(frame, "服务器缓冲区已满！", "错误", JOptionPane.ERROR\_MESSAGE);

return;// 结束线程

} else {// 普通消息

msgList.append(message + "\r\n");

}

} catch (IOException e) {

e.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

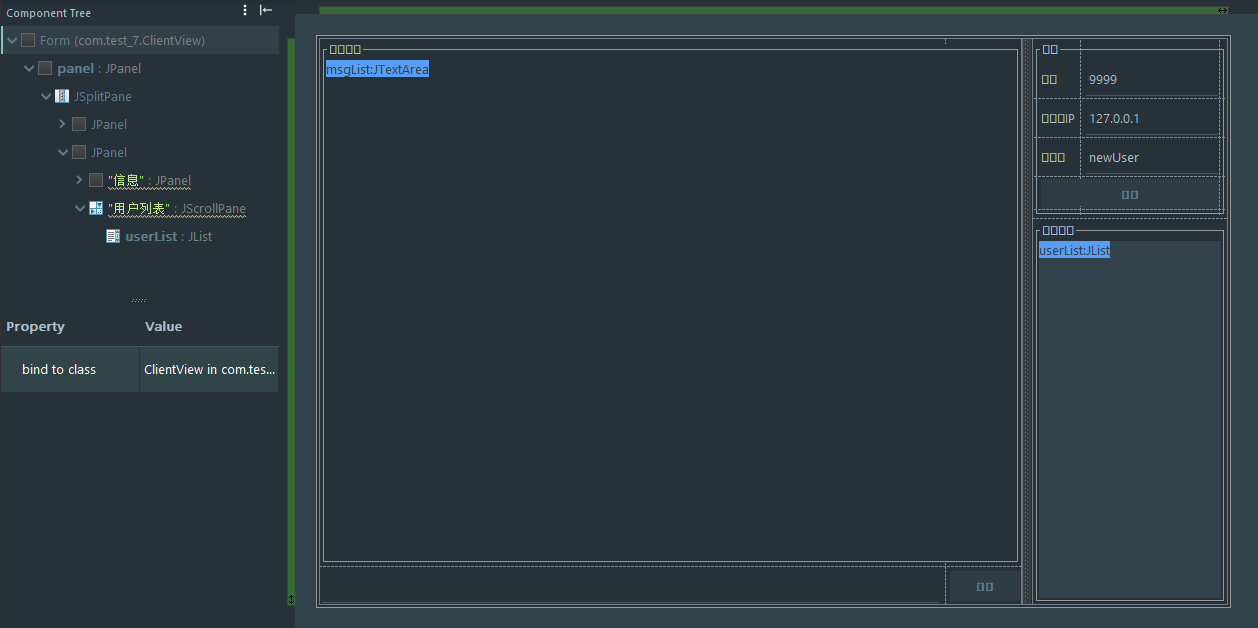
}

}

}

}

客户端界面设计采用JavaSwing结合IDEA开发工具的可视化布局设计。效果如图。布局中的各种空间对应Swing的各种对象，布局完成后通过绑定控件函数完成具体的业务功能。



### 2.2.3 Tensorflow聊天机器人的实现

（1）数据预处理

机器人训练数据来自于网上下载的数据。主要是一些电影中的对话数据，原始数据比较乱,无法操作，需要进行数据的预处理，数据预处理主要包括分词、词性标注等工作。通过Batch类完成工作：

创建batch类，里面包含了encoder输入，decoder输入，decoder标签，decoder样本长度mask

class Batch:

def \_\_init\_\_(self):

self.encoderSeqs = []

self.decoderSeqs = []

self.targetSeqs = []

self.weights = []

#一个加载数据集的方法

def loadDataset(filename):

dataset\_path = os.path.join(filename)

print('Loading dataset from {}'.format(dataset\_path))

with open(dataset\_path, 'rb') as handle:

data = pickle.load(handle)

word2id = data['word2id']

id2word = data['id2word']

trainingSamples = data['trainingSamples']

return word2id, id2word, trainingSamples

#创建batch

def createBatch(samples, en\_de\_seq\_len):

batch = Batch()

#根据样本长度获得batch size大小

batchSize = len(samples)

#将每条数据的问题和答案分开传入到相应的变量中

for i in range(batchSize):

sample = samples[i]

batch.encoderSeqs.append(list(reversed(sample[0]))) # 将输入反序，可提高模型效果

batch.decoderSeqs.append([goToken] + sample[1] + [eosToken]) # Add the <go> and <eos> tokens

batch.targetSeqs.append(batch.decoderSeqs[-1][1:]) # Same as decoder, but shifted to the left (ignore the <go>)

# 将每个元素PAD到指定长度，并构造weights序列长度mask标志

batch.encoderSeqs[i] = [padToken] \* (en\_de\_seq\_len[0] - len(batch.encoderSeqs[i])) + batch.encoderSeqs[i]

batch.weights.append([1.0] \* len(batch.targetSeqs[i]) + [0.0] \* (en\_de\_seq\_len[1] - len(batch.targetSeqs[i])))

batch.decoderSeqs[i] = batch.decoderSeqs[i] + [padToken] \* (en\_de\_seq\_len[1] - len(batch.decoderSeqs[i]))

batch.targetSeqs[i] = batch.targetSeqs[i] + [padToken] \* (en\_de\_seq\_len[1] - len(batch.targetSeqs[i]))

encoderSeqsT = [] # Corrected orientation

for i in range(en\_de\_seq\_len[0]):

encoderSeqT = []

for j in range(batchSize):

encoderSeqT.append(batch.encoderSeqs[j][i])

encoderSeqsT.append(encoderSeqT)

batch.encoderSeqs = encoderSeqsT

decoderSeqsT = []

targetSeqsT = []

weightsT = []

for i in range(en\_de\_seq\_len[1]):

decoderSeqT = []

targetSeqT = []

weightT = []

for j in range(batchSize):

decoderSeqT.append(batch.decoderSeqs[j][i])

targetSeqT.append(batch.targetSeqs[j][i])

weightT.append(batch.weights[j][i])

decoderSeqsT.append(decoderSeqT)

targetSeqsT.append(targetSeqT)

weightsT.append(weightT)

batch.decoderSeqs = decoderSeqsT

batch.targetSeqs = targetSeqsT

batch.weights = weightsT

return batch

#使用ntlk进行分词、词性标注等工作。

def getBatches(data, batch\_size, en\_de\_seq\_len):

#每个epoch之前都要进行样本的shuffle

random.shuffle(data)

batches = []

data\_len = len(data)

def genNextSamples():

for i in range(0, data\_len, batch\_size):

yield data[i:min(i + batch\_size, data\_len)]

for samples in genNextSamples():

batch = createBatch(samples, en\_de\_seq\_len)

batches.append(batch)

return batches

def sentence2enco(sentence, word2id, en\_de\_seq\_len):

if sentence == '':

return None

#分词

tokens = nltk.word\_tokenize(sentence)

if len(tokens) > en\_de\_seq\_len[0]:

return None

#将每个单词转化为id

wordIds = []

for token in tokens:

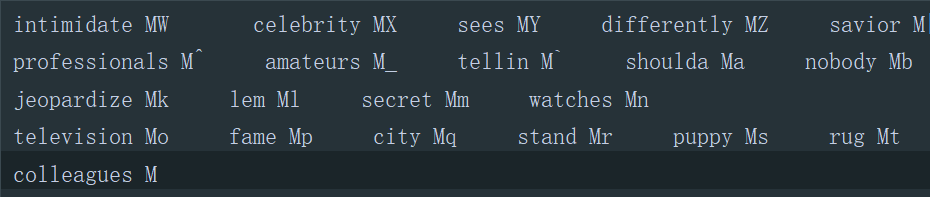
wordIds.append(word2id.get(token, unknownToken))

#调用createBatch构造batch

batch = createBatch([[wordIds, []]], en\_de\_seq\_len)

return batch

处理后的数据大概如下：



（2）模型构建

对原始数据处理后接下在需要构建模型。使用的是Seq2Seq模型。

def create\_model(session, forward\_only, beam\_search, beam\_size=5):

"""创建转换模型并在Session中初始化或加载参数。"""

model = Seq2SeqModel(

FLAGS.en\_vocab\_size, FLAGS.en\_vocab\_size, [10, 10],

FLAGS.size, FLAGS.num\_layers, FLAGS.batch\_size,

FLAGS.learning\_rate, forward\_only=forward\_only, beam\_search=beam\_search, beam\_size=beam\_size)

ckpt = tf.train.latest\_checkpoint(FLAGS.train\_dir)

model\_path = 'tmp\chat\_bot.ckpt-0' #模型保存路径

if forward\_only:

model.saver.restore(session, model\_path)

elif ckpt and tf.gfile.Exists(ckpt.model\_checkpoint\_path):

print("Reading model parameters from %s" % ckpt.model\_checkpoint\_path)

model.saver.restore(session, ckpt.model\_checkpoint\_path)

else:

print("Created model with fresh parameters.")

session.run(tf.initialize\_all\_variables())

return model

（3）模型训练

对模型进行训练，这个过程需要等待很长时间，训练出的模型好坏除了取决于选用的模型还取决于数据的质量及数据多少。

def train():

data\_path = 'data\dataset-cornell-length10-filter1-vocabSize40000.pkl' #数据集路径

word2id, id2word, trainingSamples = loadDataset(data\_path)

with tf.Session() as sess:

model = create\_model(sess, False, beam\_search=False, beam\_size=5)

current\_step = 0

for e in range(FLAGS.numEpochs):

batches = getBatches(trainingSamples, FLAGS.batch\_size, model.en\_de\_seq\_len)

for nextBatch in tqdm(batches, desc="Training"):

\_, step\_loss = model.step(sess, nextBatch.encoderSeqs, nextBatch.decoderSeqs, nextBatch.targetSeqs,

nextBatch.weights, goToken)

current\_step += 1

if current\_step % FLAGS.steps\_per\_checkpoint == 0:

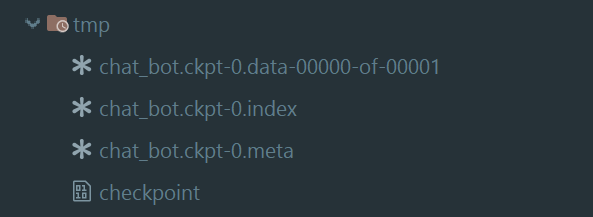
perplexity = math.exp(float(step\_loss)) if step\_loss < 300 else float('inf')

tqdm.write("----- Step %d -- Loss %.2f -- Perplexity %.2f" % (current\_step, step\_loss, perplexity))

checkpoint\_path = os.path.join(FLAGS.train\_dir, "chat\_bot.ckpt")

model.saver.save(sess, checkpoint\_path, global\_step=model.global\_step)

训练出的模型回保存在当前目录的tmp目录下：



（4）模型测试

编写测试代码对模型进程测试，检验模型的效果。

def train():

with tf.Session() as sess:

beam\_size = FLAGS.beam\_size

beam\_search = FLAGS.beam\_search

model = create\_model(sess, True, beam\_search=beam\_search, beam\_size=beam\_size)

model.batch\_size = 1

data\_path = 'data\dataset-cornell-length10-filter1-vocabSize40000.pkl'

word2id, id2word, trainingSamples = loadDataset(data\_path)

if beam\_search:

sys.stdout.write("YOU> ")

sys.stdout.flush()

sentence = sys.stdin.readline()

while sentence:

batch = sentence2enco(sentence, word2id, model.en\_de\_seq\_len)

beam\_path, beam\_symbol = model.step(sess, batch.encoderSeqs, batch.decoderSeqs, batch.targetSeqs,

batch.weights, goToken)

paths = [[] for \_ in range(beam\_size)]

curr = [i for i in range(beam\_size)]

num\_steps = len(beam\_path)

for i in range(num\_steps - 1, -1, -1):

for kk in range(beam\_size):

paths[kk].append(beam\_symbol[i][curr[kk]])

curr[kk] = beam\_path[i][curr[kk]]

recos = set()

print("AI>", end="")

for kk in range(beam\_size):

foutputs = [int(logit) for logit in paths[kk][::-1]]

if eosToken in foutputs:

foutputs = foutputs[:foutputs.index(eosToken)]

rec = " ".join([tf.compat.as\_str(id2word[output]) for output in foutputs if output in id2word])

if rec not in recos:

recos.add(rec)

print(rec)

print("YOU> ", end="");

sys.stdout.flush()

sentence = sys.stdin.readline()

（5）构建模型服务

确保模型可用之后，搭建AI服务给服务端使用。具体代码如下：

import socket

import threading

import json

import chat as cb

def main():

# 创建服务器套接字

serversocket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# 获取本地主机名称

host = socket.gethostname()

# 设置一个端口

port = 9998

# 将套接字与本地主机和端口绑定

serversocket.bind((host, port))

# 设置监听最大连接数

serversocket.listen(5)

# 获取本地服务器的连接信息

myaddr = serversocket.getsockname()

print("服务器地址:%s" % str(myaddr))

# 循环等待接受客户端信息

while True:

# 获取一个客户端连接

clientsocket, addr = serversocket.accept()

print("连接地址:%s" % str(addr))

try:

t = ServerThreading(clientsocket) # 为每一个请求开启一个处理线程

t.start()

pass

except Exception as identifier:

print(identifier)

pass

pass

serversocket.close()

pass

class ServerThreading(threading.Thread):

# words = text2vec.load\_lexicon()

def \_\_init\_\_(self, clientsocket, recvsize=1024 \* 1024, encoding="utf-8"):

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

self.\_socket = clientsocket

self.\_recvsize = recvsize

self.\_encoding = encoding

pass

def run(self):

print("开启线程.....")

try:

while True:

print("等待接受消息")

msg = ''

while True:

# 读取recvsize个字节

rec = self.\_socket.recv(self.\_recvsize)

# 解码

msg += rec.decode(self.\_encoding)

if msg.strip().endswith('over'):

msg = msg[:-4]

break

print("收到消息：", msg)

# 调用模型处理请求

res = "AI@ALL@" + cb.chat(msg)

sendmsg = json.dumps(res)+"\n"

# 发送数据

self.\_socket.send(("%s" % sendmsg).encode(self.\_encoding))

print(sendmsg+" 已发送！")

except Exception as identifier:

self.\_socket.send("500".encode(self.\_encoding))

print(identifier)

pass

finally:

self.\_socket.close()

print("任务结束.....")

pass

def \_\_del\_\_(self):

pass

if \_\_name\_\_ == "\_\_main\_\_":

main()

# 系统测试

## 命名管道服务端客户端测试

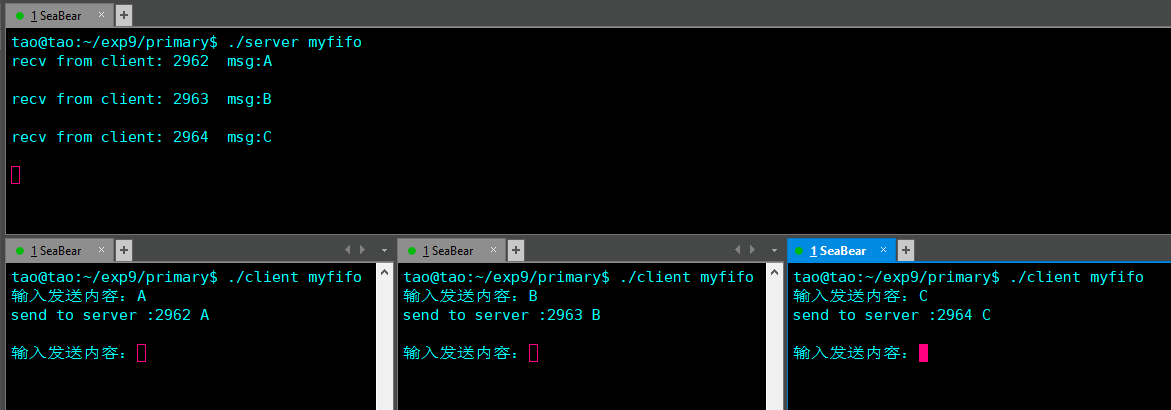


图3-1 命名管道测试

可以看出多个服务端端可以允许多个客户端发送请求。

## 多人聊天测试

启动服务端：

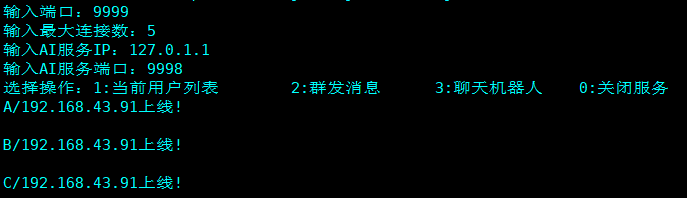


图 3-2 聊天服务器启动

A、B、C三个用户聊天如下：

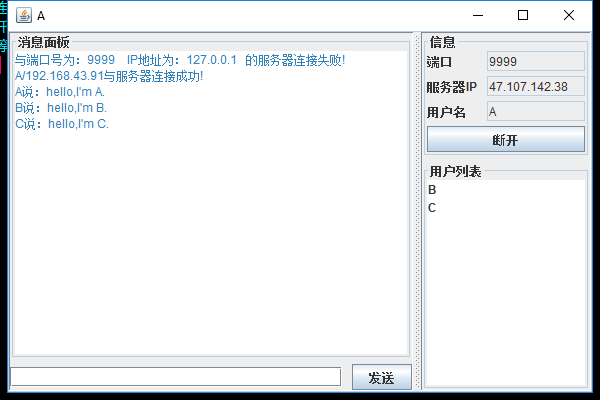


图 3-3 客户端A

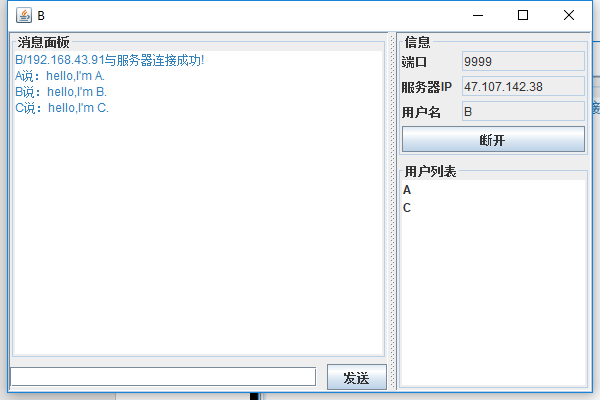


图 3-4 客户端B

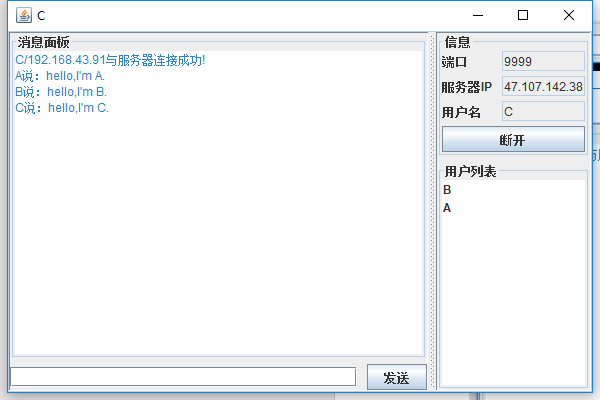


图 3-5 客户端C

测试结果可以看出，多人聊天的基本功能已经实现，服务器使用的公网IP，所以可以实现不同主机间聊天。但是功能相对来说还比较单一。

## AI聊天测试

启动AI服务

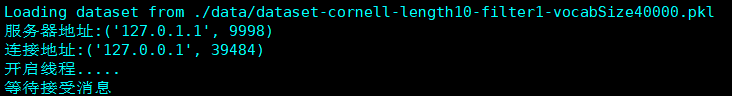


图 3-6 启动AI服务

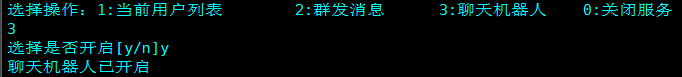


图 3-7 开启聊天机器人

聊天测试

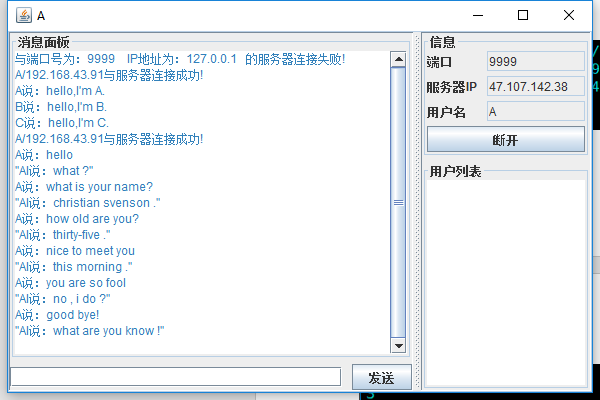


图 3-8 与机器人对话

测试结果一般。虽然基本功能达到了，但是机器人还是很傻。

# 总结

本次综合实验主要学习了命名管道、socket、tensorflow及相关应用的开发。综合设计总体围绕着C/S架构去开发简单的应用。先是使用了命名管道实现了多客户端请求服务端，深入明白了命名管道的机制，更加熟悉了与其相关的系统调用的使用。中级题目使用socket网络编程技术实现了基于TCP的网络聊天室，正好这个学期学习了计算机网络，其中也学习了网络通信及socket简单编程，在明白socket通信的原理之后，可以简单的实验客户端和服务端的数据交流。但是要实现多人在线聊天，仅仅使用socket是不够的，需要引入多线程技术。编程语言大部分都比较支持多线程，但使用的难易及开发难度是不同的，相比C语言的多线程，Java、python等高级语言要方便很多，最终选择了Java语言进行开发，使用Java语言的另一个好处是他是跨平台的编写一套代码即可在windows下运行又可以在linux下运行。使用多线程可以很好的实现多人聊天但需要注意的是使用多线程之前要设计好，不然很有可能越来越乱，特别是当服务停止时，会出现很多错误。使用Java开发出客户端和服务端之后可以把客户端放在服务器上运行，这样只要通过连接主机端口便可访问服务，实现不同电脑之间的通信。

高级题目做的tensorflow环境搭建以及实现简单的聊天机器人，做高级题目耗费了很多时间，不是因为太难无从下手，而是遇到了太多的坑。从开始的搭建tensorflow环境开始，就遇到了各种版本不兼容问题，比如最开始安装的是Anaconda3 5.3.0它的python版本是3.7.x。安装完成后使用pip安装tensorflow发现没有任何版本可以安装，查了一下才发现是python3.7.x太新了没有兼容。Tensorflow有两个版本tensorflow-gpu和tensorflow-cpu，使用GPU版本需要CUDA等依赖，由于种种原因导致GPU版本最终没有安装成功，只安装了tensorflow-cpu，但CPU版本比GPU版本慢很多。在环境安装好之后，使用tensorflow训练模型又遇到了问题，新版本的tensorflow比起老版本变化了很多，导致很多老版本的代码无法运行报错。最后一切没问题后就是漫长的训练模型时间。

总的来说，这次实验收获还是很多的，在一次次踩坑之后，也记住了那些问题的解决办法。从多线程编程到深度学习模型训练，巩固了自己的编程能力，虽然最后模型的效果不太好但是也是一次不错的进步。以前线程、进程、配置各种环境等都很懵，学习完linux之后对他们不管从原理上还是从实践上都有了不错的理解和操作，可以将自己掌握的知识更加联系在一起，对于计算机操作系统的理解也开了一个不错的头。

# 参考文献

（1）李武波, 张蕾, 舒鑫. 基于Seq2Seq的生成式自动问答系统应用与研究[J]. 现代计算机(专业版), 2017(36).

（2）王倩, 邓媛劼. 基于Java Socket的网络聊天系统的设计[J]. 电脑与信息技术, 2018(1).

（3）参考连接：http://www.tensorfly.cn/ 2019年1月2日