目录

[1 功能模块说明 2](#_Toc23044)

[1.1 闲聊模块 2](#_Toc23338)

[1.2 面部识别模块 2](#_Toc8630)

[1.3 心理咨询模块 3](#_Toc23305)

[1.4 心情记录模块 4](#_Toc1887)

[2 系统运行流程图与环境说明 5](#_Toc15331)

[2.1 系统运行流程图 5](#_Toc27250)

[2.2 系统运行环境说明 5](#_Toc20575)

[3 系统核心代码 6](#_Toc5876)

[3.1 日常闲聊模块核心代码 6](#_Toc1733)

[3.1.1 客户端UI核心代码 6](#_Toc17478)

[3.1.2 服务器端核心代码 19](#_Toc7511)

[3.2 面部识别模块核心代码 42](#_Toc29715)

[3.2.1 客户端UI核心代码 42](#_Toc17925)

[3.2.2 服务器端核心代码 50](#_Toc6639)

[3.3 心理咨询模块核心代码 54](#_Toc7263)

[3.4 心情记录模块核心代码 64](#_Toc18480)

# **功能模块说明**

* 1. 闲聊模块

用户与聊天机器人进行简单的闲谈对话交互。用户从客户端输入文本，客户端获取并且转发用户聊天内容，向服务器端发送对话文本，服务器端接收处理后向客户端返回回复内容，客户端接收回复，并显示在界面上。

闲聊模块运行截图如图1-1所示。

图1-1 闲聊模块运行截图

* 1. 面部识别模块

用户可以从图库中选择一张照片或者在APP中使用相机实时拍摄面部，客户端储存图片信息，发送到服务器端。服务器端检测人脸是否存在、并加载面部情绪模型识别面部表情，然后将情绪返回至客户端。客户端根据返回的情绪内容作出不同的反应。

面部识别模块运行截图如图1-2所示。

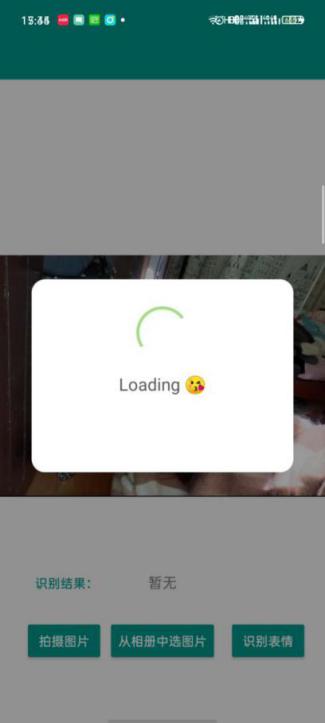
  

图1-2 面部情绪识别模块运行截图

* 1. 心理咨询模块

用户可在该界面上选择需要疏导的问题方向，客户端根据用户的选择，作出不同的判断，并引导用户，提出解决方案。同时，根据用户每日心情的不同，该模块功能会作出不同的反应。如，用户今日心情为负面状态，则用户进入该功能界面时，系统自己弹出“讲个笑话”选项，帮助分散用户注意力，舒缓心情。

心理咨询模块运行截图如图1-3所示。

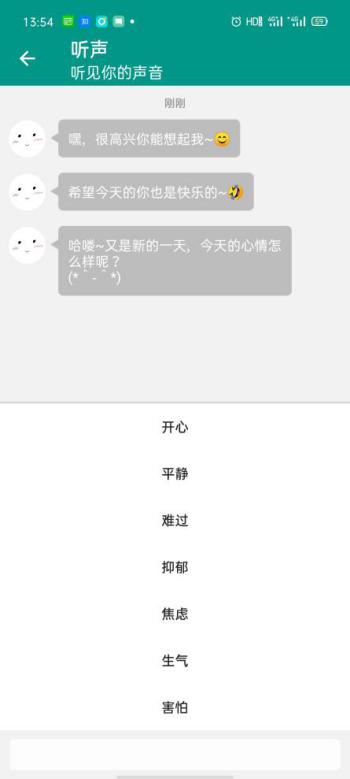
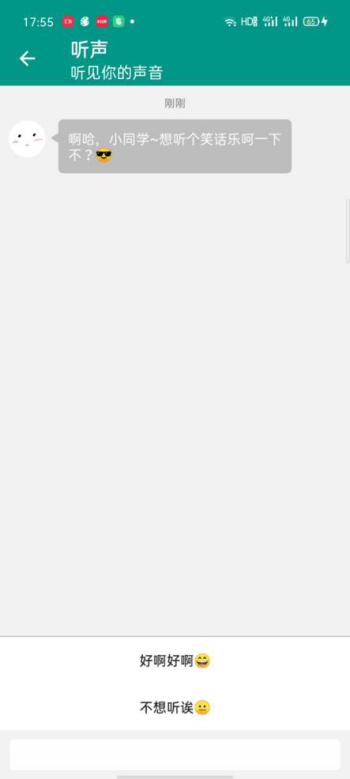
  

图1-3 心理咨询模块运行截图

* 1. 心情记录模块

每日实时记录用户的心情状态，并绘制成心情记录图反馈给用户，让用户对于自己的心理变化有一个清楚、直观的了解。

心情记录模块运行截图如图1-4所示。

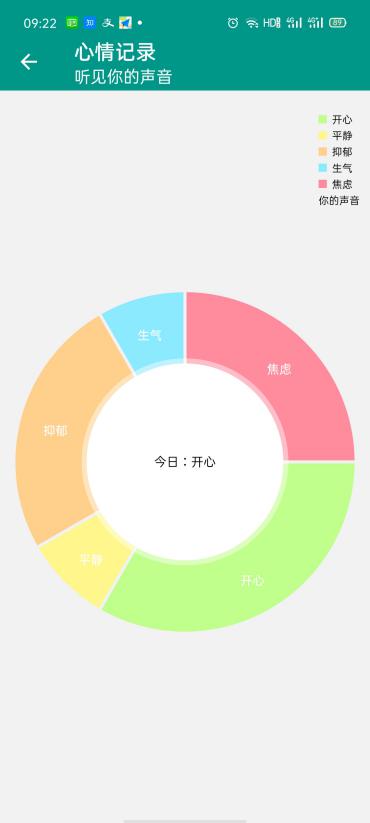


图1-4 心情记录模块运行截图

# **系统运行流程图与环境说明**

* 1. 系统运行流程图

系统功能运行流程图如图2-1所示。

运行系统

选择模式

闲聊模式

心理咨询模式

面部识别模式

心情记录模式

用户输入文本并发送

客户端发送至服务器端

服务器端接收处理，返回结果

客户端更新UI，显示结果

用户拍摄面部/选择图片并发送

当日心情是否记录

用户选择心情，系统记录

用户选择心理咨询内容

客户端系统提供疏导方法

用户查看心情记录

图2-1 系统运行流程图

* 1. 系统运行环境说明

1. 硬件环境

本地开发机：ASUS笔记本。64位操作系统Win10家庭版。机带RAM8.00GB。英特尔i7-6500U CPU处理器。

Android客户端：Oppo Reno 3元气版真机。Android 10.0操作系统。

1. 软件环境

表2-1 软件环境配置表

|  |  |  |
| --- | --- | --- |
| **软件名称** | **版本** | **说明** |
| PyCharm | 2019.3.1 | Python开发环境 |
| Android Studuio | 3.1.2 | Android开发环境 |
| Python | 3.6.5 | 服务器开发 |
| JDK | 1.8.0 | JAVA开发工具 |

# 系统核心代码

* 1. 日常闲聊模块核心代码
     1. 客户端UI核心代码

|  |
| --- |
| 客户端Activity |
| protected void onCreate(Bundle savedInstanceState) { // 新建函数  super.onCreate(savedInstanceState);   setContentView(R.layout.activity\_main); //初始化页面  ButterKnife.bind(this);  initData(); // 初始化数据  initView(); // 初始化视图  initListener(); // 初始化监听事件  initIP(); // 初始化IP   //initGifView();  }  // 初始化数据  private void initData() {  if (msgList.size() == 0) { // 第一句提示对话设置  MessageEntity entity = new MessageEntity(ChatMessageAdapter.TYPE\_LEFT, TimeUtil.getCurrentTimeMillis());  entity.setText("嗨嗨~来聊天吧！");  msgList.add(entity);  }  msgAdapter = new ChatMessageAdapter(this, msgList); // 对话适配器  lvMessage.setAdapter(msgAdapter); // 设置为机器人对话  lvMessage.setSelection(msgAdapter.getCount()); }  private void initView() {  toolbar.setTitle(getString(R.string.app\_name)); // 设置标题名  setSupportActionBar(toolbar); //设置标题栏 }  // 初始化监听事件  private void initListener() {   ivSendMsg.setOnClickListener(onClickListener); // 发送按钮  lvMessage.setOnScrollListener(new AbsListView.OnScrollListener() { // 滚动条  @Override  public void onScrollStateChanged(AbsListView view, int scrollState) {  KeyBoardUtil.hideKeyboard(mActivity); // 隐藏键盘  }   @Override  public void onScroll(AbsListView view, int firstVisibleItem, int visibleItemCount, int totalItemCount) {  }  });  }  //用户输入，view为当前活动视图 public void setRight(View view) {  String msg = etMsg.getText().toString().trim(); //用户聊天文本  if (!IsNullOrEmpty.isEmpty(msg)) { //判断内容是否为空  MessageEntity entity = new MessageEntity(ChatMessageAdapter.TYPE\_RIGHT, TimeUtil.getCurrentTimeMillis(), msg); //存储消息的实体  msgList.add(entity); //消息列表增加  msgAdapter.notifyDataSetChanged(); //消息显示适配器更改  sendMsg = etMsg.getText().toString(); //发送的消息内容，etMsg为聊天输入框实体  SendSocket(view); //调用发送消息函数  etMsg.setText(""); //更新聊天输入框  } }  @Override public boolean onOptionsItemSelected(MenuItem item) { // Item项选择时间  super.onOptionsItemSelected(item); // 继承  switch (item.getItemId()) { // 不同的Item ID  case R.id.action\_about: // 相关信息  NavigateManager.gotoAboutActivity(mContext);  return true;  case R.id.action\_help: // 心理咨询  NavigateManager.gotoHelpActivity(mContext);  return true;  case R.id.action\_mood: // 心情记录  NavigateManager.gotoMoodActivity(mContext);  return true;  case R.id.action\_socket: // socket   NavigateManager.gotoSocketActivity(mContext);  return true;  case R.id.action\_face: // 面部识别  NavigateManager.gotoFaceActivity(mContext);  return true;  default:  return false;  } }  //处理UI更新  private Handler handler = new Handler() {  @Override  public void handleMessage(Message msg) { /// 对于接收到的Message 消息  super.handleMessage(msg);  switch (msg.what) { //选择message类型  case 1:  setLeft(msg.obj.toString()); // 更新用户输入消息  break;  case 2:  setLeft("网络连接失败！"); // 网络失败提示  break;  }  }  };  public void loading() {  /\*\*  \* "加载项"布局，此布局被添加到ListView的Footer中。  \*/  LinearLayout mLoadLayout = new LinearLayout(this);  mLoadLayout.setMinimumHeight(60);  mLoadLayout.setGravity(Gravity.CENTER);  mLoadLayout.setOrientation(LinearLayout.HORIZONTAL);  /\*\*  \* 向"加载项"布局中添加一个圆型进度条。  \*/  ProgressBar mProgressBar = new ProgressBar(this);  mProgressBar.setPadding(0, 0, 15, 0);  mLoadLayout.addView(mProgressBar);  /\*\*  \* 向"加载项"布局中添加提示信息。  \*/  TextView mTipContent = new TextView(this);  mTipContent.setText("加载中...");  mLoadLayout.addView(mTipContent);  /\*\*  \* 获取ListView组件，并将"加载项"布局添加到ListView组件的Footer中。  \*/  lvMessage.addFooterView(mLoadLayout); }  //机器人回答,msg 为聊天信息 public void setLeft(String msg) {   MessageEntity entity = new MessageEntity(); //存储消息的实体  entity.setTime(TimeUtil.getCurrentTimeMillis()); //设置时间  entity.setType(ChatMessageAdapter.TYPE\_LEFT); //设置类型为机器人类型  entity.setText(msg); //初始化消息内容  msgList.add(entity); //消息列表增加  msgAdapter.notifyDataSetChanged(); //消息适配器更新 }  //发送消息与接收方法  public void SendSocket(View view) {  new Thread() {  @Override  public void run() {  try {  Socket sed\_socket = new Socket(); //新的套接字  SocketAddress socAddress = new InetSocketAddress(IP, PORT); //设置套接字地址  sed\_socket.connect(socAddress, 5000); //进行socket连接  boolean connected = sed\_socket.isConnected(); //socket连接状态  OutputStream os = sed\_socket.getOutputStream(); //输出流  PrintWriter pw = new PrintWriter(os);  String msg = sendMsg; //消息文本  pw.write(msg); //输出信息  os.flush(); //刷新缓冲区  pw.close();  Socket rev\_socket = new Socket(IP, PORT); //接收的socket  try {  BufferedReader in = new BufferedReader(new InputStreamReader(rev\_socket.getInputStream(), "UTF-8")); //缓冲区读取类型  String s = in.readLine(); //读缓冲区  Message message = handler.obtainMessage(); //Message作为反馈UI的信息  message.what = 1; //反馈类型设置  message.obj = s; //反馈内容  handler.sendMessage(message); // handler更新UI  in.close();  } catch (IOException e) {  Log.i("连接", e.getMessage().toString()); //错误信息  }   } catch (IOException e) {  Log.i("连接", e.getMessage().toString()); //错误信息  e.printStackTrace(); //打印错误  Message message = handler.obtainMessage();  message.what = 2;  handler.sendMessage(message); //UI处理错误显示   }  }  }.start(); }  //检测用户今日心情状态  public String JudgeSad(){  DataBase dbHelper = new DataBase(this, "Chatbot.db", null, 1); //数据库  SQLiteDatabase db = dbHelper.getReadableDatabase();// 打开数据库  String sql = "select \* from MoodRecord where date = '"+getMyDate()+"' and mood = '开心'";  Cursor cursor = db.rawQuery(sql, null); //执行SQL语句  if(cursor.getCount()==0) {cursor.close();db.close();return "1";} //判断情绪是否负面  else {cursor.close();db.close();return "0";} } |
| 客户端UI |
| <LinearLayout // 线性布局  xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:background="@color/bk"  android:orientation="vertical">  <include layout="@layout/app\_base\_toolbar"/>  <RelativeLayout  android:layout\_width="match\_parent"  android:layout\_height="match\_parent">  <ListView // 对话页面  android:id="@+id/lv\_message"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:layout\_above="@+id/divider"  android:divider="#00000000"  android:dividerHeight="0px"  android:transcriptMode="normal"/>  <View // 分割线  android:id="@+id/divider"  android:layout\_width="match\_parent"  android:layout\_height="1dp"  android:layout\_above="@+id/rl\_msg"  android:background="@color/font\_black\_5"/>  <RelativeLayout // 对话框布局  android:id="@+id/rl\_msg"  android:layout\_width="fill\_parent"  android:layout\_height="50dp"  android:layout\_alignParentBottom="true"  android:gravity="center\_vertical">  <ImageView //打底色  android:id="@+id/iv\_send\_msg"  android:layout\_width="30dp"  android:layout\_height="30dp"  android:layout\_alignParentRight="true"  android:layout\_centerVertical="true"  android:layout\_marginRight="10dp"  android:src="@drawable/selector\_send"/>  <LinearLayout // 对话框布局  android:id="@+id/ll\_msg"  android:layout\_width="match\_parent"  android:layout\_height="32dp"  android:layout\_centerVertical="true"  android:layout\_marginLeft="10dp"  android:layout\_marginRight="10dp"  android:layout\_toLeftOf="@+id/iv\_send\_msg"  android:background="@drawable/actionsheet\_single\_normal"  android:orientation="vertical">  <EditText // 对话框  android:id="@+id/et\_msg"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:layout\_gravity="center\_vertical"  android:background="@color/white"  android:hint="输入您的问题"  android:imeOptions="actionSend|flagNoEnterAction"  android:inputType="textCapSentences|textAutoCorrect|textMultiLine|textShortMessage"  android:maxLength="100"  android:maxLines="4"  android:minHeight="32dp"  android:paddingLeft="10dp"  android:paddingRight="10dp"  android:textColor="@color/font\_black\_2"  android:textColorHint="@color/font\_black\_5"  android:textSize="14sp"/>  </LinearLayout>  </RelativeLayout>  </RelativeLayout>  </LinearLayout> |
| 对话适配器 |
| public class ChatMessageAdapter extends BaseListAdapter<MessageEntity> { // 继承BaseListAdapter类   private Context mContext; //活动内容   public static final int TYPE\_LEFT = 0; // 机器人消息类型  public static final int TYPE\_RIGHT = 1; // 用户消息类型  // 构造函数  public ChatMessageAdapter(Context context, List<MessageEntity> list) {  super(context, list);  mContext = context;  }   @Override  public int getViewTypeCount() { // 消息类型数量  return 2;  }  // 获取消息类型  @Override  public int getItemViewType(int position) {  if (getItem(position).getType() == TYPE\_LEFT) {  return TYPE\_LEFT;  }  return TYPE\_RIGHT;  }  // 创建弹窗  private View createViewByType(int position) {  if (getItem(position).getType() == TYPE\_LEFT) {  return mInflater.inflate(R.layout.item\_conversation\_left, null);  }  return mInflater.inflate(R.layout.item\_conversation\_right, null);  }   @Override  public View getView(int position, View convertView, ViewGroup parent) {  if (convertView == null) {  convertView = createViewByType(position);  }   final MessageEntity entity = getItem(position); // 消息实体   TextView tvTime = ViewHolder.get(convertView, R.id.tv\_time);  BubbleTextVew btvMessage = ViewHolder.get(convertView, R.id.btv\_message);  // 时间设置  if (isDisplayTime(position)) {  tvTime.setVisibility(View.VISIBLE);  tvTime.setText(TimeUtil.friendlyTime(mContext, entity.getTime()));  } else {  tvTime.setVisibility(View.GONE);  } // 消息实体类型  switch (entity.getCode()) {  case TulingParams.TulingCode.URL:  btvMessage.setText(SpecialViewUtil.getSpannableString(entity.getText(), entity.getUrl()));  break;  case TulingParams.TulingCode.NEWS:  btvMessage.setText(SpecialViewUtil.getSpannableString(entity.getText(), "点击查看"));  break;  default:  btvMessage.setText(entity.getText());  break;  } // 点击事件  btvMessage.setOnClickListener(v -> {  switch (entity.getCode()) {  case TulingParams.TulingCode.URL:  NavigateManager.gotoDetailActivity(mContext, entity.getUrl());  break;  case TulingParams.TulingCode.NEWS:  NavigateManager.gotoNewsActivity(mContext, entity);  break;  }  }); // 点击事件  btvMessage.setOnLongClickListener(v -> {  copyDeleteDialog(mContext, entity);  return false;  });   return convertView;  }   // 一分钟内的请求与回复不显示时间  public boolean isDisplayTime(int position) {  if (position > 0) {  if ((getItem(position).getTime() - getItem(position-1).getTime()) > 60 \* 1000) {  return true;  } else {  return false;  }  } else if (position == 0) {  return true;  } else {  return false;  }  } // 删除、复制  private void copyDeleteDialog(Context context, MessageEntity entity) {  new MaterialDialog.Builder(context)  .items("复制该文本", "删除这一条")  .itemsCallback((dialog, view, which, text) -> {  switch (which) {  case 0:  ClipboardManager cm = (ClipboardManager) context.getSystemService(Context.CLIPBOARD\_SERVICE);  cm.setText(entity.getText());  Toast.makeText(context, "已复制", Toast.LENGTH\_SHORT).show();  break;  case 1:  getData().remove(entity);  notifyDataSetChanged();  break;  }  })  .show();  } } |
| BaseListAdapter |
| public abstract class BaseListAdapter<E> extends BaseAdapter {   private List<E> mList = new ArrayList<E>(); // 消息列表  protected Context mContext; // 活动内容  protected LayoutInflater mInflater; // 布局   public BaseListAdapter(Context context) { // 构造函数  mContext = context;  mInflater = LayoutInflater.from(mContext);  }   public BaseListAdapter(Context context, List<E> list) { // 构造函数  this(context);  mList = list;  }   @Override  public int getCount() { // 获取消息列表数量  return mList.size();  }   public void clearAll() { // 清理所有  mList.clear();  }   public void setData(List<E> list) { //设置数据  clearAll();  addALL(list);  }   public List<E> getData() { // 获取数据  return mList;  }   public void addALL(List<E> list){ //增加  if(list == null || list.size() == 0){  return ;  }  mList.addAll(list);  }  public void add(E item){ //添加一项  mList.add(item);  }   @Override  public E getItem(int position) { // 获取某个消息  return (E) mList.get(position);  }   @Override  public long getItemId(int position) { // 获取某个消息的id  return position;  }   public void removeEntity(E e){ //删除消息  mList.remove(e);  } |
| SQLite数据库 |
| public class DataBase extends SQLiteOpenHelper { // sql 新建数据库表文本  public static final String CREATE\_MoodRecord = "create table MoodRecord ("  + "mood string ,"  + "date string)"; // 心情记录  public static final String CREATE\_ChatRecord = "create table ChatRecord ("  + "seq string ,"  + "owner bool,"  + "date string)"; // 聊天记录  public static final String CREATE\_UserMessage = "create table UserMessage("  + "ip string)"; //用户数据  private Context mContext; // 构造函数  public DataBase(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {  super(context, name, factory, version);  mContext = context;  }   @Override  public void onCreate(SQLiteDatabase db) {  db.execSQL(CREATE\_MoodRecord); // 执行sql语句  db.execSQL(CREATE\_ChatRecord);  db.execSQL(CREATE\_UserMessage);  // Toast.makeText(mContext,"Create succeed",Toast.LENGTH\_SHORT).show();  } // 更新数据表  @Override  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  db.execSQL("drop table if exists MoodRecord");  db.execSQL("drop table if exists UserMessage");  Toast.makeText(mContext,"Update",Toast.LENGTH\_SHORT).show();  onCreate(db);   } } |
| 导航栏管理 |
| public class NavigateManager {  // 相关信息栏  public static void gotoAboutActivity(Context context) {  Intent intent = new Intent(context, AboutActivity.class);  context.startActivity(intent);  } // 心理咨询活动跳转  public static void gotoHelpActivity(Context context) {  Intent intent = new Intent(context, HelpActivity.class);  context.startActivity(intent);  } // 心情记录活动跳转  public static void gotoMoodActivity(Context context) {  Intent intent = new Intent(context, MoodActivity.class);  context.startActivity(intent);  }  //测试页面  public static void gotoSocketActivity(Context context) {  Intent intent = new Intent(context, SocketActivity.class);  context.startActivity(intent);  }  // 人脸识别跳转  public static void gotoFaceActivity(Context context) {  Intent intent = new Intent(context, FaceActivity.class);  context.startActivity(intent);  } } |
| 消息类实体 |
| public class MessageEntity implements Serializable {   private int type; // 类型  private long time; // 时间  private int code; // 代码  private String text; / 文本  private String url; // 连接   private List<NewsEntity> list; // 消息列表   public MessageEntity() { //构造函数  }   public MessageEntity(int type, long time) { // 构造函数  this.type = type; // 类型初始化  this.time = time; // 时间  this.text = text; //文本内容  }   public MessageEntity(int type, long time, String text) {// 构造函数  this.type = type;  this.time = time;  this.text = text;  }   public List<NewsEntity> getList() { // 获取列表  return list;  }   public void setList(List<NewsEntity> list) { // 设置列表  this.list = list;  }   public long getTime() { // 获取时间  return time;  }   public void setTime(long time) { // 设置时间  this.time = time;  }   public int getCode() {  return code; // 获取代码  }   public void setCode(int code) {  this.code = code; // 设置代号  }   public String getText() { // 获取消息  return text;  }   public void setText(String text) { // 设置消息  this.text = text;  }   public String getUrl() {   return url;  }   public void setUrl(String url) {  this.url = url;  }   public int getType() {  return type; // 获取类型  }   public void setType(int type) {  this.type = type; // 设置类型  }   @Override  public String toString() { // 转换为string  return "MessageEntity{" +  "type=" + type +  ", time=" + time +  ", code=" + code +  ", text='" + text + '\'' +  ", url='" + url + '\'' +  ", list=" + list +  '}';  } } |

* + 1. 服务器端核心代码

|  |
| --- |
| 服务器socket运行 |
| # 服务器socket运行函数  def socket\_service():  try:  s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # 新的socket  # 防止socket server重启后端口被占用（socket.error: [Errno 98] Address already in use）  s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  s.bind(('192.168.1.5', 6666))  s.listen(10) # 建立连接  except socket.error as msg: # 处理错误  print(msg)  sys.exit(1)  print('Waiting connection...')  # 运行等待连接  while 1:  conn, addr = s.accept() # 接收连接成功的目标与地址  t = threading.Thread(target=deal\_data, args=(conn, addr)) # 子进程运行deal\_data  t.start()  # 处理接收信息  def deal\_data(conn, addr):  global msg, cnt, answer # msg: 消息文本，cnt：计数，answer：回复文本  print('Accept new connection from {0}'.format(addr))  if cnt == 0: # 接收消息  data = conn.recv(1024) # 缓冲区内容存入data  print('{0} client send data is {1}'.format(addr, data.decode()))  msg = data.decode()  else: # 发送回复  answer = deal\_txt(msg) # 生成回复文本  if answer == "" or answer is None or answer == 'a' or len(clean\_zh\_text(msg)) == 0:  answer = "哎，我太笨了听不懂" # 回复文本内容无效  conn.send(answer.encode()) # 发送回复文本  cnt = (cnt+1) % 2  conn.close()  # 处理接收到的信息  def deal\_txt(msg):  answer = chat\_answer(msg) # 调用chat\_answer处理  return answer  if \_\_name\_\_ == '\_\_main\_\_':  socket\_service() |
| 文本内容处理 |
| class info(object):  def \_\_init\_\_(self, r, w, h, pt):  self.words = w # 词  self.pos\_tags = pt # 词性  self.head = h # 首部  self.relation = r # 关系  # 存父子节点的索引  self.sbv = [] # 主谓  self.vob = [] # 动宾  self.att = [] # 定中  self.adv = [] # 状中  self.com = [] # 动补 说明动作的状态、结果、程度等  self.hed = [] # 核心关系  self.coo = [] # 并列1  self.pob = [] # 并列2  self.attr\_dict\_list = [] # 关系列表  self.seq = "" # 信息文本  # 存主谓宾结构  def extract\_attr(self, r\_, w\_, h\_):  pre\_attribute\_index = -1 # 前置索引  for index, z in enumerate(zip(r\_, w\_, h\_)):  if z[0] == "SBV":  self.sbv.append((index, z[2] - 1)) # index是主语的索引，h是谓语的索引  elif z[0] == "VOB":  self.vob.append((index, z[2] - 1)) # index是宾语的索引，h是谓语的索引  elif z[0] == "ATT":  self.att.append((index, z[2] - 1)) # index是定语的索引，h是宾语的索引  elif z[0] == "ADV":  self.adv.append((index, z[2] - 1)) # index是状语的索引，h是谓语的索引  elif z[0] == "CMP":  self.com.append((index, z[2] - 1)) # index是补语的索引，h是谓语的索引（新增）  elif z[0] == "HED":  self.hed.append((index, z[2] - 1)) # index是动词的索引, h是root(-1)  elif z[0] == "COO":  self.coo.append((index, z[2] - 1)) # index是下一个并列词的索引， h是这个词的索引  elif z[0] == "POB":  self.coo.append((index, z[2] - 1)) # index是并列词的索引， h是和的索引  # 生成occ需要计算的1至多组句法成分  if self.sbv is not None:  # 主谓  for sbv\_subject\_index, sbv\_predicate\_index in self.sbv:  attr\_dict = {"subject": [self.words[sbv\_subject\_index], -1000],  "predicate": [[self.words[sbv\_predicate\_index]], -1000],  "adverbial": [[], -1000], "object": [[], -1000],  "sub\_attribute": [[], -1000], "obj\_attribute": [[], -1000],  "obj\_complement":[[],-1000],  "complement": [[], -1000], "adverbial\_com": [[], -1000]} # "complement"和"com\_adverbial"为补充  #（补充）寻找额外的谓语  for index, z in enumerate(zip(r\_, w\_, h\_)):  if z[0] == "COO" and self.words[z[2] - 1] in attr\_dict["predicate"][0]:  attr\_dict["predicate"][0].append(self.words[index])  # 状语  for adv\_adverbial\_index, adv\_predicate\_index in self.adv:  if sbv\_predicate\_index == adv\_predicate\_index:  attr\_dict["adverbial"][0].append(self.words[adv\_adverbial\_index])  for vob\_object\_index, adv\_predicate\_index in self.vob:  if sbv\_predicate\_index == vob\_object\_index:  attr\_dict["adverbial"][0].append(self.words[adv\_predicate\_index])  # 补充（补语）  for pre\_attribute\_index, predicate\_index in self.com:  if predicate\_index == sbv\_predicate\_index:  attr\_dict["complement"][0].append(self.words[pre\_attribute\_index])  # 补充(补语对应的状语)  for adv\_adverbial\_index\_1, adv\_predicate\_index\_1 in self.adv:  if adv\_predicate\_index\_1 == pre\_attribute\_index:  attr\_dict["adverbial\_com"][0].append(self.words[adv\_adverbial\_index\_1])  # 定语  for att\_attribute\_index, att\_subject\_index in self.att:  if sbv\_subject\_index == att\_subject\_index:  attr\_dict["sub\_attribute"][0].append(self.words[att\_attribute\_index])  # 找宾语  for vob\_object\_index, vob\_predicate\_index in self.vob:  if sbv\_predicate\_index == vob\_predicate\_index:  attr\_dict["object"][0].append(self.words[vob\_object\_index])  # 定语  for att\_attribute\_index, att\_object\_index in self.att:  if att\_object\_index == vob\_object\_index:  attr\_dict["obj\_attribute"][0].append(self.words[att\_attribute\_index])  for object\_com, object\_index in self.com:  if object\_index == vob\_object\_index:  attr\_dict["obj\_complement"][0].append(self.words[object\_com])  break  self.attr\_dict\_list.append(attr\_dict)  def main\_process(txt, MOOD):  generator = analysis\_sentence(txt) # 句型分析  for word, head\_num\_list, pos\_tag, relation\_list in generator:  s1 = info(relation\_list, word, head\_num\_list, pos\_tag) # 处理文本的语法信息  s1.extract\_attr(relation\_list, word, head\_num\_list) # 存储文本的语法信息  flag\_answer = 0 # 是否匹配  answer = "" # 回复内容  s1.seq = txt # 初始化文本类的原聊天信息  flag\_answer = rule\_judge(s1, MOOD) # 进行规则匹配  if flag\_answer == 0: # 规则匹配失败，进入生成式对话  if answer == 0 or answer is None or answer == "" or len(clean\_zh\_text(answer)) == 0:  answer = emotion\_polarity(s1, txt) # 先进入情绪极性回复  if answer == 0 or answer is None or answer == "" or len(clean\_zh\_text(answer)) == 0:  answer = reply(s1.seq) # 进入生成式对话内容，并去除多余文本  answer = answer.rstrip('。').rstrip('~').rstrip('？').rstrip('！').rstrip('，')  if len(answer) > 10:  answer = "我没听明白欸"  else:  answer = flag\_answer  return answer |
| 语料库文本预处理 |
| #将训练集的数据识别读取并存入一个List  convs = [] # 用于存储对话的列表  with open(conv\_path,'r',encoding='utf-8') as f:  one\_conv = [] # 存储一次完整对话  for line in f:  line = line.strip('\n').replace('/', '')#去除换行符，并将原文件中已经分词的标记去掉，重新用结巴分词.  if line == '':  continue  if line[0] == gConfig['e']:  if one\_conv:  convs.append(one\_conv)  one\_conv = []  elif line[0] == gConfig['m']:  one\_conv.append(line.split(' ')[1])#将一次完整的对话存储下来  #训练集的对话进行分类，分为问和答，作为encoder和decoder的熟练数据  seq = []  for conv in convs:  if len(conv) == 1:  continue  if len(conv) % 2 != 0:  conv = conv[:-1]  for i in range(len(conv)):  if i % 2 == 0:  conv[i]=" ".join(jieba.cut(conv[i]))#使用jieba  conv[i+1]=" ".join(jieba.cut(conv[i+1]))  seq.append(conv[i]+'\t'+conv[i+1])  seq\_train = open(gConfig['seq\_data'],'w',encoding='utf-8') # 获取存储路径  for i in range(len(seq)):  seq\_train.write(seq[i]+'\n') # 写入文件    if i % 1000 == 0:  print(len(range(len(seq))), '处理进度：', i)  seq\_train.close() |
| 训练模型与预测函数 |
| #定义一个字典用于接收配置文件的配置参数  gConfig = {}  gConfig = getConfig.get\_config(config\_file='seq2seq.ini')  vocab\_inp\_size = gConfig['enc\_vocab\_size'] #输入语句字典维度  vocab\_tar\_size = gConfig['dec\_vocab\_size'] #输出语句字典维度  embedding\_dim = gConfig['embedding\_dim'] #embedding维度  units = gConfig['layer\_size'] #层级的神经元数量  BATCH\_SIZE = gConfig['batch\_size'] #批量大小  #输入语句(问句)的最大长度、输出语句(答句)的最大长度  max\_length\_inp, max\_length\_tar = 20, 20    #定义一个语句的处理函数，在所有的语句开头和结尾分别加上'start '/' end'  def preprocess\_sentence(w):  w = 'start ' + w + ' end'  #打印信息：start 好 的 end  # print("完整句子:",w.encode("GBK","ignore").decode("GBK"))  return w    #定义一个训练数据集的处理函数，用于读取文件中的数据，并进行初步的语句处理，在所有的语句开头和结尾分别加上'start '/' end'  def create\_dataset(path, num\_examples):  #使用分行符'\n'进行分割，分割出完整的每行句子  lines = io.open(path, encoding='UTF-8').read().strip().split('\n')  #使用分行符'\t'进行分割，分割出问句和答句两部分的句子，并在所有的语句开头和结尾分别加上'start '/' end'  word\_pairs = [[preprocess\_sentence(w) for w in l.split('\t')] for l in lines[:num\_examples]]  #返回初步处理好的数据  return zip(\*word\_pairs)  #计算最大的句子长度  def max\_length(tensor):  return max(len(t) for t in tensor)    #定义 word2vec函数，通过统计训练集数据中所有字符出现频率，并且构建一个字典存储所有字符和其对应编码值，  #并使用字典中的编码值对训练集中的语句中的字符进行替换。  def tokenize(lang):  #enc\_vocab\_size构建输入语句的字典维度。配置不在字典中的字符的替换数字，一般使用“3”这个数字来代替在字典中找不到的字符。  #Tokenizer对象.index\_word[index]: 获取单词（字符串），根据传入单词所在字典中的索引值来获取该单词。  #Tokenizer对象.word\_index[word]: 获取单词在字典中的索引值，根据传入单词，获取单词所在字典中的索引值。  lang\_tokenizer = tf.keras.preprocessing.text.Tokenizer(num\_words=gConfig['enc\_vocab\_size'], oov\_token=3)  # word\_index可以返回每个被分割的单词在字典中的索引值，从1算起。仅在调用fit\_on\_texts之后设置。  lang\_tokenizer.fit\_on\_texts(lang)  # texts\_to\_sequences返回每个被分割的单词在字典中的索引值，从1算起。  # 每个句子对应一个列表，每个列表中元素值为该句子中的单词在字典中的索引值。  tensor = lang\_tokenizer.texts\_to\_sequences(lang)  # 每个句子所转换为列表，如果列表中单词所对应的索引值数量不满maxlen，则默认补0，可指定padding='post'在后面做填充  tensor = tf.keras.preprocessing.sequence.pad\_sequences(tensor, maxlen=max\_length\_inp, padding='post')  return tensor, lang\_tokenizer    #num\_examples读取训练数据的最大值  def read\_data(path, num\_examples):  # 定义一个训练数据集的处理函数，用于读取文件中的数据，并进行初步的语句处理，在所有的语句开头和结尾分别加上'start '/' end'  input\_lang, target\_lang = create\_dataset(path, num\_examples)  # print(input\_lang[:7]) #取出问句的前8个句子，分别对应答句的前8个句子  # print(target\_lang[:7]) #取出答句的前8个句子，分别对应问句的前8个句子    #tokenize函数实现了把输入语句(问句)和目标语句(答句)都通过word2vec转换为数值  #返回 输入语句(问句)的word2vec转换(每个数值均为单词在字典中的索引值)、输入语句(问句)的Tokenizer对象input\_token  #input\_tensor/target\_tensor中的每个列表对应一个句子，每个列表中元素值为该句子中的单词在字典中的索引值。  input\_tensor, input\_token = tokenize(input\_lang)  # 返回 目标语句(答句)的word2vec转换(每个数值均为单词在字典中的索引值)、目标语句(答句)的Tokenizer对象target\_token  target\_tensor, target\_token = tokenize(target\_lang)  return input\_tensor, input\_token, target\_tensor, target\_token    #seq\_data处理后的中文训练集，max\_train\_data\_size读取训练数据的最大值  input\_tensor, input\_token, target\_tensor, target\_token = read\_data(gConfig['seq\_data'], gConfig['max\_train\_data\_size'])    #训练模型  def train():  print("Preparing data in %s" % gConfig['train\_data'])  input\_tensor\_train,input\_tensor\_val,target\_tensor\_train,target\_tensor\_val = train\_test\_split(input\_tensor,target\_tensor,test\_size=0.2)  #训练数据除以批量大小：计算一个epoch循环需要训练多少步才能将训练数据训练完整一遍，steps\_per\_epoch值为312步  steps\_per\_epoch = len(input\_tensor\_train) // gConfig['batch\_size']  print("steps\_per\_epoch:",steps\_per\_epoch) #312    #计算需要随机打乱排序的数据大小BUFFER\_SIZE，将数据集随机打乱可以防止模型过多陷入局部最优解中  BUFFER\_SIZE = len(input\_tensor\_train)  #将训练数据集随机打乱  dataset = tf.data.Dataset.from\_tensor\_slices((input\_tensor\_train, target\_tensor\_train)).shuffle(BUFFER\_SIZE)  #drop\_remainder=True表示最后一个batch的数量达不到batch\_size的话则会被丢弃  dataset = dataset.batch(BATCH\_SIZE, drop\_remainder=True)    # 初始化模型保存路径  checkpoint\_dir = gConfig['model\_data']  # 初始化模型文件的保存前缀  checkpoint\_prefix = os.path.join(checkpoint\_dir, "ckpt")  #如果保存路径下已经存在预训练好的模型文件的话则直接加载  ckpt = tf.io.gfile.listdir(checkpoint\_dir)  if ckpt:  print("reload pretrained model")  #重新加载预应变模型  seq2seqModel.checkpoint.restore(tf.train.latest\_checkpoint(checkpoint\_dir))  #计算训练开始时间  start\_time = time.time()    while True:  # 计算当前训练开始时间  start\_time\_epoch = time.time()  total\_loss = 0  #对encoder编码器的隐藏层的状态进行零初始化，shape为(批量大小batch\_size 128, 神经元数量layer\_size 256)  enc\_hidden = seq2seqModel.encoder.initialize\_hidden\_state()    #批量从训练数据中取出数据进行训练  # dataset.take(312)会获取312次批量大小的数据进行遍历，遍历的值batch从0到steps\_per\_epoch-1，即从0到312-1  for (batch, (inp, targ)) in enumerate(dataset.take(steps\_per\_epoch)):  print("batch:",batch) #遍历的值从0到steps\_per\_epoch-1，即从0到312-1  print("inp.shape:",inp.shape)#(128, 20) 即(批量大小128, 输入语句(问句)的最大长度20)  print("targ.shape:",targ.shape) #(128, 20) 即(批量大小128, 输出语句(答句)的最大长度20)  #获取每步所训练的批量数据的loss值：传入 批量输入语句(问句)、批量输出语句(答句)、目标语句(答句)的Tokenizer对象、编码器隐藏层状态  batch\_loss = seq2seqModel.train\_step(inp, targ, target\_token, enc\_hidden)  #计算一个epoch的总loss值total\_loss  total\_loss += batch\_loss  print("batch\_loss:",batch\_loss.numpy())  print("total\_loss:",total\_loss.numpy())    #计算一个epoch中每步训练批量数据所要消耗的平均时间  step\_time\_epoch = (time.time() - start\_time\_epoch) / steps\_per\_epoch  #计算一个epoch中每步训练批量数据的平均loss值  step\_loss = total\_loss / steps\_per\_epoch  #计算当前已经执行的总的训练步数，即训练了多少个批量数据  current\_steps =+ steps\_per\_epoch  #计算当前已经执行的总的训练步数的其中每步的平均耗时，即训练每个批量数据的平均耗时  step\_time\_total = (time.time() - start\_time) / current\_steps  #每一个epoch结束后打印一次信息  print('训练总步数: {} 每步耗时: {} 最新epoch中的每步耗时: {} 最新epoch中的每步loss值 {:.4f}'.  format(current\_steps, step\_time\_total, step\_time\_epoch, step\_loss.numpy()))  #把每一个epoch结束后训练好的模型文件进行保存  # seq2seqModel.checkpoint.save(file\_prefix=checkpoint\_prefix)  #刷新命令行输出  sys.stdout.flush()    #预测模型  def predict(sentence):  # 定义一个语句的处理函数，对输入语句的开头和结尾分别加上'start '/' end'  sentence = preprocess\_sentence(sentence)  #对输入语句的每个字符进行word2vec转换，返回每个字符在字典中的所对应的索引值  inputs = [input\_token.word\_index.get(i, 3) for i in sentence.split(' ')]  #对输入语句进行进行word2vec转换后的句子按照maxlen最大长度进行以0补全，默认补0，可指定padding='post'在后面做填充  inputs = tf.keras.preprocessing.sequence.pad\_sequences([inputs], maxlen=max\_length\_inp, padding='post')  #将输入语句转换为tensor  inputs = tf.convert\_to\_tensor(inputs)    # 初始化模型保存路径  checkpoint\_dir = gConfig['model\_data']  # 重新加载预应变模型  seq2seqModel.checkpoint.restore(tf.train.latest\_checkpoint(checkpoint\_dir))    #初始化输出变量  result = ''  #初始化隐藏层  hidden = [tf.zeros((1, units))]  #使用编码器encoder对输入语句(问句inputs)、隐藏层状态hidden进行提取特征，最后返回编码器输出enc\_out和编码器隐藏层状态enc\_hidden  enc\_out, enc\_hidden = seq2seqModel.encoder(inputs, hidden)  #初始化解码器的隐藏层：编码器隐藏层状态enc\_hidden作为解码器隐藏层状态dec\_hidden  dec\_hidden = enc\_hidden  #初始化解码器的输入：定义答句的开头首单词为“'start'在字典中的”索引值作为解码器的输入语句  #Tokenizer.word\_index[word] 获取单词在字典中的索引值，根据传入单词，获取单词所在字典中的索引值  dec\_input = tf.expand\_dims([target\_token.word\_index['start']], 0)    #开始按照输出语句(答句)的最大长度进行预测，即遍历的最大次数为输出语句(答句)的最大长度，但如果解码器输出的单词为'end'即结束遍历循环  for t in range(max\_length\_tar):  #根据输入信息，逐字对输出语句(答句)进行预测，解码器最终返回预测值、解码器隐藏层状态、注意力权重  # 第一次预测：传入上一个单词'start'、编码器隐藏层状态、编码器输出  # 第一次后面的预测：传入上一个时间步预测出的单词、上一个时间步的解码器隐藏层状态、编码器输出  predictions, dec\_hidden, attention\_weights = seq2seqModel.decoder(dec\_input, dec\_hidden, enc\_out)  #获取预测的结果：argmax获取向量(列表)中最大元素值所在的索引值  predicted\_id = tf.argmax(predictions[0]).numpy()  #Tokenizer.index\_word[wordIndex] 获取单词（字符串），根据传入单词所在字典中的索引值来获取该单词  #根据所预测的索引值通过字典获取对应的单词，如果解码器输出的单词为'end'(结束标识)即结束遍历循环停止预测  if target\_token.index\_word[predicted\_id] == 'end':  break  #输出语句(答句)仍然还没有结束，继续在结尾追加空格  result += target\_token.index\_word[predicted\_id] + ' '  #在第一层增加维度，shape变成(1,1)，继续把所预测的索引值作为上文输入信息然后加入到解码器中来预测下一个单词的数值  dec\_input = tf.expand\_dims([predicted\_id], 0)  #返回所预测的完整输出语句(答句)  return result |
| Seq2Seq模型 |
| gConfig = {}  gConfig = getConfig.get\_config(config\_file='seq2seq.ini')    class Encoder(tf.keras.Model):  def \_\_init\_\_(self, vocab\_size, embedding\_dim, enc\_units, batch\_sz):  super(Encoder, self).\_\_init\_\_()  self.batch\_sz = batch\_sz #批量大小为128  self.enc\_units = enc\_units #网络层神经元数量为256  # 配置Embedding层：编码器字典大小为20000、embedding维度为128  self.embedding = tf.keras.layers.Embedding(vocab\_size, embedding\_dim)  # 配置GRU层：神经元数量为256  # return\_sequences：配置是否在输出的句子中返回最后的输出数据。  # return\_state：配置是否将训练的最后状态添加到输出数据中返回。  # recurrent\_initializer：配置循环网络核的初始化权重矩阵，用于对循环神经元的状态进行线性变换。  self.gru = tf.keras.layers.GRU(self.enc\_units, return\_sequences=True, return\_state=True, recurrent\_initializer='glorot\_uniform')    #传入 批量输入语句(问句)、编码器隐藏层状态 ，返回 编码器输出、编码器隐藏层状态输出  def call(self, x, hidden):  #使用Embedding层对批量输入语句(问句)进行编码，返回shape为(128, 20, 128)，即(批量大小128, 句子最大长度20, embedding维度128)的数据  x = self.embedding(x)  print("Encoder x.shape:",x.shape) #(128, 20, 128)  #将进行编码过后的数据输入到GRU层，最终返回shape为(128, 20, 256)，即(批量大小128, 句子最大长度20, GRU层神经元数量256)的编码器输出、  # 还有shape为(128, 256)，即(批量大小128, GRU层神经元数量256)的编码器隐藏层状态输出  output, state = self.gru(x, initial\_state=hidden)  print("Encoder output.shape:",output.shape)#(128, 20, 256)  print("Encoder state.shape:",state.shape) #(128, 256)  return output, state    #创建编码器的隐藏层状态  def initialize\_hidden\_state(self):  #对encoder编码器的隐藏层的状态进行零初始化，shape为(批量大小batch\_size 128, 神经元数量layer\_size 256)  return tf.zeros((self.batch\_sz, self.enc\_units))    class BahdanauAttention(tf.keras.Model):  # 传入 网络层神经元数量为256  def \_\_init\_\_(self, units):  super(BahdanauAttention, self).\_\_init\_\_()  self.W1 = tf.keras.layers.Dense(units) #配置Dense全连接层的神经元数量为256  self.W2 = tf.keras.layers.Dense(units) #配置Dense全连接层的神经元数量为256  self.V = tf.keras.layers.Dense(1) #配置Dense全连接层的神经元数量为1  # 调用Attention注意力机制类的call方法：传入 编码器隐藏层状态/上一个时间步解码器隐藏层状态、编码器输出  # 返回 上下文向量context\_vector、注意力权重attention\_weights  def call(self, query, values):  # 首先是 hidden的shape为(batch\_size, hidden size) 即 (批量大小128, 神经元数量256)，  # 然后转换为 hidden\_with\_time\_axis的shape为(batch\_size, 1, hidden size)即 (批量大小128, 1, 神经元数量256)。  hidden\_with\_time\_axis = tf.expand\_dims(query, 1)  print("BahdanauAttention hidden\_with\_time\_axis.shape:",hidden\_with\_time\_axis.shape) #(128, 1, 256)    # 计算分数score，最终shape为(128, 20, 1)，即(batch\_size, max\_length句子最大长度20, hidden\_size)，Dense层V的神经元数量hidden\_size为1  # values实际为 shape为(批量大小128, 句子最大长度20, GRU层神经元数量256)的编码器输出enc\_output  score = self.V(tf.nn.tanh(self.W1(values) + self.W2(hidden\_with\_time\_axis)))  print("BahdanauAttention score.shape:",score.shape) #(128, 20, 1)    # 注意力权重attention\_weights的shape为(128, 20, 1) 即(batch\_size, max\_length句子最大长度20, 1)  attention\_weights = tf.nn.softmax(score, axis=1)  print("BahdanauAttention attention\_weights.shape:",attention\_weights.shape)#(128, 20, 1)    #上下文向量context\_vector = 注意力权重(128, 20, 1) \* 编码器输出(128, 20, 256)  # 最终context\_vector的shape为(128, 20, 256) 即(批量大小128, 句子最大长度20, GRU层神经元数量256)  context\_vector = attention\_weights \* values  print("BahdanauAttention context\_vector.shape:",context\_vector.shape) #(128, 20, 256)    # 上下文向量context\_vector经过sum之后的shape为(batch\_size, hidden\_size) 即 (批量大小128, GRU层神经元数量256)  context\_vector = tf.reduce\_sum(context\_vector, axis=1)  print("BahdanauAttention reduce\_sum.shape:",context\_vector.shape) #(128, 256)  # 返回 上下文向量context\_vector、注意力权重attention\_weights  return context\_vector, attention\_weights      class Decoder(tf.keras.Model):  # 传入 解码器字典大小、embedding维度、网络层神经元数量、批量大小 初始化 Decoder解码器  def \_\_init\_\_(self, vocab\_size, embedding\_dim, dec\_units, batch\_sz):  super(Decoder, self).\_\_init\_\_()  self.batch\_sz = batch\_sz #批量大小为128  self.dec\_units = dec\_units #网络层神经元数量为256  # 配置Embedding层：解码器字典大小为20000、embedding维度为128  self.embedding = tf.keras.layers.Embedding(vocab\_size, embedding\_dim)  # 配置GRU层：神经元数量为256  # return\_sequences：配置是否在输出的句子中返回最后的输出数据。  # return\_state：配置是否将训练的最后状态添加到输出数据中返回。  # recurrent\_initializer：配置循环网络核的初始化权重矩阵，用于对循环神经元的状态进行线性变换。  self.gru = tf.keras.layers.GRU(self.dec\_units, return\_sequences=True, return\_state=True, recurrent\_initializer='glorot\_uniform')  #配置Dense全连接层的神经元数量为解码器字典大小为20000，即作为最后一层的分类器  self.fc = tf.keras.layers.Dense(vocab\_size)  #创建初始化注意力机制对象：传入 网络层神经元数量为256  self.attention = BahdanauAttention(self.dec\_units)    # 调用decoder解码器的call方法：传入 单词'start'/下一个时间步的真实单词、编码器隐藏层状态/上一个时间步解码器隐藏层状态、编码器输出  # 解码器最终返回 当前时间步的解码器预测输出值、解码器隐藏层状态、注意力权重  def call(self, x, hidden, enc\_output):  #调用Attention注意力机制类的call方法：传入 编码器隐藏层状态/上一个时间步解码器隐藏层状态、编码器输出  # 返回 上下文向量context\_vector、注意力权重attention\_weights  context\_vector, attention\_weights = self.attention(hidden, enc\_output)  #使用Embedding层对单词'start'/下一个时间步的真实单词进行编码，返回shape为(批量大小128, 1, embedding维度128)的数据  x = self.embedding(x)  print("Decoder x.shape:",x.shape) #(128, 1, 128)    #上下文向量context\_vector的shape为(batch\_size, hidden\_size)，即 (批量大小128, GRU层神经元数量256)，  #经过拓增后shape为(batch\_size, 1, hidden\_size) 即 (批量大小128, 1, GRU层神经元数量256)。  #然后对(批量大小128, 1, GRU层神经元数量256)的上下文向量context\_vector 和 (批量大小128, 1, embedding维度128)的x 两者在最后一个维度的进行concat，  #最终concat的结果的shape为(128, 1, 384)。  x = tf.concat([tf.expand\_dims(context\_vector, 1), x], axis=-1)  print("Decoder concat.shape:",x.shape) #(128, 1, 384)    #将进行编码过后的数据输入到GRU层，因为GRU层神经元数量hidden\_size为256，  #因此 GRU层最终返回 shape为(128, 1, 256) 即(批量大小128, 1, GRU层神经元数量256)的当前时间步的解码器预测输出值、  #还有 shape为(128, 256) 即(批量大小128, GRU层神经元数量256)的解码器隐藏层状态state。  output, state = self.gru(x)  print("Decoder output.shape:",output.shape) #(128, 1, 256)  print("Decoder state.shape:",state.shape) #(128, 256)    #把当前时间步的解码器预测输出值output的shape的前两个维度进行合并为一个维度，即output的shape从(128, 1, 256)变成(128, 256) 即(批量大小128, GRU层神经元数量256)  output = tf.reshape(output, (-1, output.shape[2]))  print("Decoder reshape output.shape:",output.shape) #(128, 256)    #使用最后一层的分类器Dense全连接层进行分类，Dense层hidden\_size为解码器器字典大小20000，  # 因此最终返回当前时间步的解码器预测输出值，shape为(128, 20000) 即(批量大小128, 解码器器字典大小20000)  x = self.fc(output)  print("Decoder fc x.shape:",x.shape) #(128, 20000)    return x, state, attention\_weights    #配置字典的大小 vocabulary size，建议字典大小为20000  vocab\_inp\_size = gConfig['enc\_vocab\_size'] #编码器字典大小 20000  vocab\_tar\_size = gConfig['dec\_vocab\_size'] #解码器字典大小 20000  #配置embedding的维度为128，就是用多长的向量对单词来进行编码  embedding\_dim = gConfig['embedding\_dim']  # 配置循环神经网络层级的神经元数量  # 典型选项 : 128, 256, 512, 1024  units = gConfig['layer\_size']  BATCH\_SIZE = gConfig['batch\_size'] # 配置批量大小为128    #传入 编码器字典大小、embedding维度、网络层神经元数量、批量大小 初始化 Encoder编码器  encoder = Encoder(vocab\_inp\_size, embedding\_dim, units, BATCH\_SIZE)  #传入 解码器字典大小、embedding维度、网络层神经元数量、批量大小 初始化 Encoder编码器  decoder = Decoder(vocab\_tar\_size, embedding\_dim, units, BATCH\_SIZE)  optimizer = tf.keras.optimizers.Adam()  loss\_object = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)  checkpoint = tf.train.Checkpoint(optimizer=optimizer, encoder=encoder, decoder=decoder)      # 传入 批量输出语句(答句)中第t个时间步上的单词所在字典的索引值、当前时间步的解码器预测输出值，最终计算返回当前时间步的loss值  def loss\_function(real, pred):    #先标识出 批量输出语句(答句)中的填充值0的位置，通过equal把这些填充值0所在的位置都标识出来为True，  #同时通过logical\_not判断两者相同为True的话，则最终返回False，那么最终mask掩码中对应填充值0的位置上的值便会被置为False，  #目的是去除填充值0对最终计算loss的影响，填充值0便不会被计算其中。  mask = tf.math.logical\_not(tf.math.equal(real, 0))  #使用SparseCategoricalCrossentropy计算loss，传入 批量输出语句(答句)中第t个时间步上的单词所在字典的索引值、当前时间步的解码器预测输出值  #SparseCategoricalCrossentropy是可以接受稀疏编码的多对数交叉熵，所谓的接受稀疏编码就是指期望值可以是整型的分类编码，如1，2，3等。  loss\_ = loss\_object(real, pred)  mask = tf.cast(mask, dtype=loss\_.dtype)  #mask掩码中对应填充值0的位置上的值便会被置为False，目的是去除填充值0对最终计算loss的影响，填充值0便不会被计算其中。  loss\_ \*= mask  return tf.reduce\_mean(loss\_)      # 获取每步所训练的批量数据的loss值：传入 批量输入语句(问句)、批量输出语句(答句)、目标语句(答句)的Tokenizer对象、编码器隐藏层状态  # @tf.function  def train\_step(inp, targ, targ\_lang, enc\_hidden):  #每步所训练的批量数据的loss值  loss = 0  with tf.GradientTape() as tape:  #调用encoder编码器的call方法：传入 批量输入语句(问句)、编码器隐藏层状态 ，返回 编码器输出、编码器隐藏层状态输出  #shape为(批量大小128, 句子最大长度20, GRU层神经元数量256)的编码器输出enc\_output、shape为(批量大小128, GRU层神经元数量256)的编码器隐藏层状态输出enc\_hidden  enc\_output, enc\_hidden = encoder(inp, enc\_hidden)  #把编码器隐藏层状态输出 作为 解码器隐藏层状态的输入  dec\_hidden = enc\_hidden  #根据目标语句(答句)的Tokenizer对象.word\_index['start'] 获取该单词在字典中的索引值，shape为(批量大小128,)，  #然后又进行维度拓增 最终返回shape为(批量大小128, 1)的数据  dec\_input = tf.expand\_dims([targ\_lang.word\_index['start']] \* BATCH\_SIZE, 1)  print("dec\_input.shape:",dec\_input.shape) #(128, 1)    #批量输出语句(答句).shape[1]为输出语句(答句)的最大长度20，但不从0开始遍历而是从1开始遍历，因为不需要遍历首个单词'start'  for t in range(1, targ.shape[1]):  # 调用decoder解码器的call方法：传入 单词'start'/下一个时间步的真实单词、编码器隐藏层状态/上一个时间步解码器隐藏层状态、编码器输出  # 解码器最终返回 当前时间步预测出的单词、当前时间步解码器隐藏层状态、注意力权重。  predictions, dec\_hidden, \_ = decoder(dec\_input, dec\_hidden, enc\_output)  # targ[:, t] 获取的是批量输出语句(答句)中第t个时间步上的单词所在字典的索引值，shape为(128,) 即(批量大小128,)  # 传入 批量输出语句(答句)中第t个时间步上的单词所在字典的索引值、当前时间步的解码器预测输出值 计算当前时间步的loss值  # 先计算一个批量数据中每个时间步的loss值，然后汇总每个时间步的loss值为一个批量数据中的总loss值  loss += loss\_function(targ[:, t], predictions)  print("targ[:, t].shape:", targ[:, t].shape) #(128,)  #targ[:, t] 获取的是批量输出语句(答句)中第t个时间步上的单词所在字典的索引值，shape为(128,) 即(批量大小128,)，  #然后又进行维度拓增 最终返回shape为(128, 1) 即(批量大小128, 1)的数据，即把下一个时间步的真实单词作为解码器的输入进行下次预测  dec\_input = tf.expand\_dims(targ[:, t], 1)  print("dec\_input.shape:", dec\_input.shape) #(128, 1)    #loss为一个批量数据中每个时间步的loss值所汇总的loss值，然后除以总的时间步数(即输出语句(答句)的最大长度20)，得出一个批量数据中的平均loss值  batch\_loss = (loss / int(targ.shape[1]))  #把编码器和解码器网络的参数都作为可自动微调的参数  variables = encoder.trainable\_variables + decoder.trainable\_variables  #通过应用loss对编码器和解码器网络的参数进行求出相应的梯度  gradients = tape.gradient(loss, variables)  #根据相应的梯度对编码器和解码器网络的参数进行梯度下降  optimizer.apply\_gradients(zip(gradients, variables))  #返回一个批量数据中的平均loss值  return batch\_loss |
| 生成式对话 |
| # 定义应答函数，用于获取输入信息并返回相应的答案  def reply(req\_msg):  # 从请求中获取参数信息  # 将语句使用结巴分词进行分词  req\_msg=" ".join(jieba.cut(req\_msg))  # 调用decode\_line对生成回答信息  res\_msg = execute.predict(req\_msg)  # 如果接受到的内容为空，则给出相应的回复  if res\_msg == ' ':  res\_msg = '请与我聊聊天吧'  print(res\_msg)  return res\_msg |
| Seq2Seq基本配置 |
| [strings]  # 配置执行器的运行模式 Mode : train, test, serve  mode = train  # 处理后的中文训练集  seq\_data = train\_data/seq.data  train\_data=train\_data  # 训练集原始文件  resource\_data = train\_data/xiaohuangji50w\_nofenci.conv    #读取识别原始文件中段落和行头的标示  e = E  m = M    model\_data = model\_data    [ints]  # 配置字典的大小 vocabulary size，建议字典大小为20000  # 20000是一个合理的大小  enc\_vocab\_size = 20000  dec\_vocab\_size = 20000  #配置embedding的维度，就是用多长的向量对单词来进行编码  embedding\_dim=128    # 配置循环神经网络层级的神经元数量  # 典型选项 : 128, 256, 512, 1024  layer\_size = 256  # 配置读取训练数据的最大值，一般当显存或者内存不足时可以适当减少训练数据大小  # 数据集大小限制；通常为none：无限制  max\_train\_data\_size = 50000  # 配置批量大小  batch\_size = 128  # coding=utf-8  from configparser import SafeConfigParser    # configparser为 用于读取配置文件的包  def get\_config(config\_file='seq2seq.ini'):  parser = SafeConfigParser()  parser.read(config\_file,encoding="utf-8")  #获取int、float、string等类型的参数，按照key-value形式保存  \_conf\_ints = [(key, int(value)) for key, value in parser.items('ints')]  # \_conf\_floats = [ (key, float(value)) for key,value in parser.items('floats') ]  \_conf\_strings = [(key, str(value)) for key, value in parser.items('strings')]  #封装为一个字典对象，包含所读取的所有参数  return dict(\_conf\_ints + \_conf\_strings) |
| 结构判断文本积极性 |
| SBV = "SBV"  ADV = "ADV"  CMP = "CMP"  VOC = 'VOC'  HED = 'HED'  COO = 'COO'  def seek\_adv\_deny(adv\_list, deny\_path):  deny\_df = pd.read\_csv(deny\_path)  deny\_list = deny\_df.loc[:, 'word'].tolist()  del deny\_df  adv\_list[1]=1;  for word\_adv in adv\_list[0]:  if word\_adv in deny\_list:  adv\_list[1] = adv\_list[1] \* (-1)  # 判断修饰难 的 词 否定返回1 不否定返回 0  def judge\_adv\_deny(adv, deny\_path):  deny\_df = pd.read\_csv(deny\_path)  deny\_list = deny\_df.loc[:, 'word'].tolist()  del deny\_df  if adv in deny\_list:  return 1  else:  return 0  def judge\_specific\_sub(sub\_list, sub\_path): # 1 找到 0 没找到 从指定列表找词 是不是存在  sub\_df = pd.read\_csv(sub\_path)  my\_list = sub\_df.loc[:,"word"].tolist()  del sub\_df  sub\_list[1] = 0  for word\_sub in sub\_list[0]:  if word\_sub in my\_list:  sub\_list[1] = 1  break  if sub\_list[1] == 1:  return 1  else:  return 0  def is\_in(x, path):  df = pd.read\_csv(path)  my\_list = df.loc[:,"word"].tolist()  del df  if x in my\_list:  return 1  else:  return 0  def find\_index(TYPE, temp, word): # 找关系中对应的下标  index = []  if TYPE == SBV: # 类型判断  for sub\_temp, pre\_temp in temp.sbv:  if sub\_temp == word:  index.append(pre\_temp)  # return pre\_temp  elif TYPE == ADV:  for adv\_temp, pre\_temp in temp.adv:  if pre\_temp == word:  index.append(adv\_temp)  # return adv\_temp  elif TYPE == CMP:  for adv\_temp, pre\_temp in temp.com:  if pre\_temp == word:  index.append(adv\_temp)  elif TYPE == COO:  for adv\_temp, pre\_temp in temp.coo:  if pre\_temp == word:  index.append(adv\_temp)  return index  # 判断 x的词性 1:积极 0:中性 -1:消极  def judge\_neg\_pos(x, neg\_path, pos\_path):  neg\_df = pd.read\_csv(neg\_path)  pos\_df = pd.read\_csv(pos\_path)  pos\_list = pos\_df.loc[:, "word"].tolist()  neg\_list = neg\_df.loc[:, "word"].tolist()  del pos\_df  del neg\_df  if x in pos\_list:  return 1  elif x in neg\_list:  return -1  else:  return 0  # 找temp中第一个满足条件的下标  def find\_specific(TYPE, temp, my\_path):  my\_df = pd.read\_csv(my\_path)  my\_list = my\_df.loc[:, "word"].tolist()  del my\_df  if TYPE == SBV:  for sub\_temp, pre\_temp in temp.sbv:  if temp.words[sub\_temp] in my\_list:  return sub\_temp  elif TYPE == HED:  for hed\_temp, pre\_temp in temp.hed:  if temp.words[hed\_temp] in my\_list:  return hed\_temp  return -1  def exam\_check(obj, pos\_path, neg\_path, deny\_path, exam\_path, self\_path): # exam\_path 就是 主体的路径  temp = obj  # print(temp.words)  flag = True  tot\_emo = 1  for word in temp.words:  if is\_in(word, exam\_path) == 1:  flag = False  for emo in temp.words:  if is\_in(emo, neg\_path) == 1:  tot\_emo = -1  break  break  for word in temp.words:  if is\_in(word, deny\_path) == 1:  tot\_emo = tot\_emo \* (-1)  if flag: # 压根没有考试  return 0  # print(tot\_emo)  flag = False  attr\_dict = []  for atr in temp.attr\_dict\_list:  flag = True  attr\_dict = atr  break  if flag:  sign\_sub(attr\_dict["subject"]) # 主语是我，宾语是学业/学业  sign\_obj\_predicate(attr\_dict["predicate"], neg\_path) # 消极存在是1  exam\_sub\_id = find\_specific(SBV, temp, exam\_path)  if exam\_sub\_id == -1:  # 考砸了 ,先找 HED里有没有考，有的话直接找CMP(砸，考)||COO(za,kao)，和ADV(MEI,考)  hed\_id = find\_specific(HED, temp, exam\_path)  pre\_deny = -1  if hed\_id == -1:  pass  else: # 考砸了  adv\_id = find\_index(ADV, temp, hed\_id) # 没 考  if len(adv\_id) == 0: # 考  pass  else: # 没 考  for temp\_deny in adv\_id:  if judge\_adv\_deny(temp.words[temp\_deny], deny\_path):  pre\_deny = pre\_deny \* (-1)  pre\_id = find\_index(CMP, temp, hed\_id)  if len(pre\_id) == 0: # 考。  # 找COO  coo\_id = find\_index(COO, temp, hed\_id)  if len(coo\_id) == 0:  tmp = judge\_neg\_pos(temp.words[hed\_id], neg\_path, pos\_path)  if tmp == -1 and pre\_deny == -1: #  return -1  elif tmp == -1 and pre\_deny == 1: #  return 1  elif tmp == 1 and pre\_deny == -1: #  return 1  elif tmp == 1 and pre\_deny == 1: #  return -1  else:  return tot\_emo  else:  pre\_id = coo\_id  pre\_emotion = judge\_neg\_pos(temp.words[pre\_id[0]], neg\_path, pos\_path)  if pre\_emotion == 1 and pre\_deny == -1: # 考好  return 1  elif pre\_emotion == 1 and pre\_deny == 1: # 没考好  return -1  elif pre\_emotion == -1 and pre\_deny == -1: # 考砸  return -1  elif pre\_emotion == -1 and pre\_deny == 1: # 没考砸  return 1  else:  return tot\_emo  else: # 我（不）（觉得）考试（不）难  # if attr\_dict["subject"][1] == 4: # 考试难，考试没过  pre\_id = find\_index(SBV, temp, exam\_sub\_id) # 得到难/考  adv\_id = find\_index(ADV, temp, pre\_id[0])  cmp\_id = find\_index(CMP, temp, pre\_id[0]) # 考试没考好 的 好  cmp\_emo = 0  if len(cmp\_id) == 0:  pass  else:  cmp\_emo = judge\_neg\_pos(temp.words[cmp\_id[0]], neg\_path, pos\_path)  # 判断pre\_id极性  if len(pre\_id) == 0:  return tot\_emo  pre\_emo = judge\_neg\_pos(temp.words[pre\_id[0]], neg\_path, pos\_path)  pre\_deny = -1  if len(adv\_id) == 0: # 不存在adv  pre\_deny = -1  else: # 追溯所有的否定前缀  for temp\_deny in adv\_id:  # (temp.words[temp\_deny])  if judge\_adv\_deny(temp.words[temp\_deny], deny\_path):  pre\_deny = pre\_deny \* (-1)  # 有 我 觉得  if attr\_dict["subject"][1] == 1:  me\_id = find\_specific(SBV, temp, self\_path)  if me\_id == -1:  pass  else:  pre\_id = find\_index(SBV, temp, me\_id) # 觉得  if len(pre\_id) == 0: # 考试 难  pass  else:  adv\_id = find\_index(ADV, temp, pre\_id[0]) # 不！觉得  if len(adv\_id) == 0: # 不存在adv 我觉得 考试难  pass  else: # 我 （）觉得考试难  for temp\_deny in adv\_id:  if judge\_adv\_deny(temp.words[temp\_deny], deny\_path):  pre\_deny = pre\_deny \* (-1)  if pre\_emo == -1 and pre\_deny == -1: # 难  return -1  elif pre\_emo == -1 and pre\_deny == 1: # 不难  return 1  elif pre\_emo == 1 and pre\_deny == -1: # 简单  return 1  elif pre\_emo == 1 and pre\_deny == 1: # 不简单  return -1  else: # 一般  if pre\_emo == 0:  if pre\_deny == 1 and cmp\_emo == 1: #  return -1  elif pre\_deny == -1 and cmp\_emo == 1: #  return 1  elif pre\_deny == 1 and cmp\_emo == -1: #  return 1  elif pre\_deny == -1 and cmp\_emo == -1: #  return -1  return tot\_emo  return tot\_emo  # 我觉得 考试难  # 判断 SBV(考试存不存在) |

* 1. 面部识别模块核心代码
     1. 客户端UI核心代码

|  |
| --- |
| 面部识别 |
| public void SendSocket(View view) { // 发送图片  new Thread() {  @Override  public void run() {  try {  Socket sed\_socket = new Socket(); //新的套接字  SocketAddress socAddress = new InetSocketAddress(IP, PORT); //设置地址  sed\_socket.connect(socAddress, 5000); //连接  boolean connected = sed\_socket.isConnected(); //判断连接  Log.i("连接","连接？" + connected);  DataOutputStream out = new DataOutputStream(sed\_socket.getOutputStream()); //输出流  FileInputStream fis = new FileInputStream(imagePath); // 输出文件流  //发送图片大小  int size = fis.available();  String s = String.valueOf(size);  while(s.length()<10){  s = s + " ";  }  byte[] bytes = s.getBytes(); // 图片信息转为二进制  out.write(bytes);  out.flush();  //发送图片  //读取图片到ByteArrayOutputStream  byte[] sendBytes = new byte[1024];   int length = 0; //信息长度  while ((length = fis.read(sendBytes, 0, sendBytes.length)) > 0) {  out.write(sendBytes, 0, length); //写信息到缓冲区  out.flush();  }  fis.close();   try { //获取结果  BufferedReader in = new BufferedReader(new InputStreamReader(sed\_socket.getInputStream(), "UTF-8")); //读缓冲区工具  String emotion = in.readLine(); //读取结果信息   Message message = handler.obtainMessage(); // UI更新命令  message.what = 1; //UI更新类型  message.obj = emotion; //UI内容  handler.sendMessage(message); //handler处理  Log.i("连接", emotion);  in.close();  Log.i("连接", "下");  }catch (IOException e){ // 错误信息  Log.i("连接", "读取失败");  Log.i("连接", e.getMessage().toString());  }   } catch (IOException e) { //错误信息  Log.i("连接", "失败");  e.printStackTrace();  Log.i("连接", e.getMessage().toString());  Message message = handler.obtainMessage();  message.what = 2;  handler.sendMessage(message);  }  }  }.start();  }  private void openCa(){  Intent intent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);  startActivityForResult(intent, 0);  }  private void selectImage(){  Intent intent = new Intent(Intent.ACTION\_PICK,MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI);  startActivityForResult(intent,1);  }   @Override  protected void onActivityResult(int requestCode, int resultCode, Intent data) {  if((requestCode==0 || requestCode==1 )&&resultCode==RESULT\_OK) {  String imgPath = null;  if (requestCode == 0) {    String sdStatus = Environment.getExternalStorageState();  if (!sdStatus.equals(Environment.MEDIA\_MOUNTED)) { // 检测sd是否可用  Log.v("TestFile", "SD card is not avaiable/writeable right now.");  return;  }   Bundle bundle = data.getExtras();  Bitmap bitmap = (Bitmap) bundle.get("data");// 获取相机返回的数据，并转换为Bitmap图片格式  FileOutputStream b = null;  File file = new File("/sdcard/myImage/");  file.mkdirs();// 创建文件夹，名称为myimage   //照片的命名，目标文件夹下，以当前时间数字串为名称，即可确保每张照片名称不相同。网上流传的其他Demo这里的照片名称都写死了，则会发生无论拍照多少张，后一张总会把前一张照片覆盖。细心的同学还可以设置这个字符串，比如加上“ＩＭＧ”字样等；   String str=null;  Date date=null;  SimpleDateFormat format = new SimpleDateFormat("yyyyMMddHHmmss");//获取当前时间，进一步转化为字符串  date =new Date();  str=format.format(date);  String fileName = "/sdcard/myImage/"+str+".jpg";  imgPath = fileName;  try {  b = new FileOutputStream(fileName);  bitmap.compress(Bitmap.CompressFormat.JPEG, 100, b);// 把数据写入文件  } catch (FileNotFoundException e) {  e.printStackTrace();  } finally {  try {  b.flush();  b.close();  } catch (IOException e) {  e.printStackTrace();  }  }  }   if (requestCode == 1) {  //解析得到所选相册图片的地址  Uri selectImg = data.getData();  String[] filePathColumn = {MediaStore.Images.Media.DATA};  Cursor cursor = FaceActivity.this.getContentResolver().query(selectImg,filePathColumn,null,null,null);  cursor.moveToFirst();  int columnIndex = cursor.getColumnIndex(filePathColumn[0]);  imgPath = cursor.getString(columnIndex);  cursor.close();  }   imagePath = imgPath;  bitmap = BitmapFactory.decodeFile(imgPath);  Log.d(TAG, "bitmap: "+bitmap);  imageView.setImageBitmap(bitmap);   }  }  // 点击事件  class MyClickListener implements View.OnClickListener {  @Override  public void onClick(View v) {  // TODO Auto-generated method stub  switch (v.getId()) {  case R.id.btn\_get\_photo:  openCa();  flag = 1;  break;  case R.id.btn\_get\_photo\_from\_album:  selectImage();  flag = 2;  break;  case R.id.btn\_grey\_photo:  if(imagePath==null){  ShowErrorDialog(");  break;  }  ShowLoadingDialog("Loading );  //SendOkHttp(v);  SendSocket(v);  break;    default:  break;  }  }   }  // oncreate函数，初始化控件  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_face);  imagePath = null;  mcontext = this;  imageView = (ImageView)findViewById(R.id.img\_ccx);  getPhotoButton = (Button)findViewById(R.id.btn\_get\_photo);  greyPhotoButton = (Button)findViewById(R.id.btn\_grey\_photo);  getPhotoFromAlbum = (Button)findViewById(R.id.btn\_get\_photo\_from\_album);  resultShow = (TextView)findViewById(R.id.result);  getPhotoButton.setOnClickListener(new MyClickListener());  getPhotoFromAlbum.setOnClickListener(new MyClickListener());  greyPhotoButton.setOnClickListener(new MyClickListener());  emoji\_result = (TextView)findViewById(R.id.emoji\_result);  pDialog = new SweetAlertDialog(this, SweetAlertDialog.PROGRESS\_TYPE);   }  // UI处理  private Handler handler = new Handler() {  @Override  public void handleMessage(Message msg) {  super.handleMessage(msg);  switch (msg.what) {  case 1:  //Toast.makeText(mcontext,msg.obj.toString(),Toast.LENGTH\_LONG).show();  if(msg.obj.toString().contains("空")){  emoji\_result.setText("啊哦，没有识别到人脸);  ShowErrorDialog("没有识别到);  break;  }  ShowSuccessDialog(emojiChange(msg.obj.toString()));  emoji\_result.setText(msg.obj.toString());  String new\_alr = ChangeMood(msg.obj.toString());  InsertMood(new\_alr);  break;  case 2:  ShowErrorDialog("网络连接失败！");  break;  }  }   }; // 心情转化  public String ChangeMood(String cur){  switch (cur){  case "喜悦":  return "开心";  case "悲伤":  return "难过";  case "愤怒":  return "生气";  case "平静":  return "平静";  default:  return "难过";  }  }  public String getMyDate(){  SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy年MM月dd日");// HH:mm:ss //获取当前时间  Date date = new Date(System.currentTimeMillis());  return simpleDateFormat.format(date);  } //心情跟新  public void InsertMood(String cur) {  DataBase dbHelper = new DataBase(this, "Chatbot.db", null, 1);  SQLiteDatabase sqLiteDatabase = dbHelper.getWritableDatabase();   String sql = "update MoodRecord set mood = '"+cur+"' where date='"+getMyDate()+"'";    sqLiteDatabase.execSQL(sql);   Toast.makeText(this,"今日心情更新成功!"+getMyDate()+"?",Toast.LENGTH\_SHORT).show();  }   public void ShowLoadingDialog(String msg){  pDialog = new SweetAlertDialog(this, SweetAlertDialog.PROGRESS\_TYPE);  pDialog.getProgressHelper().setBarColor(Color.parseColor("#A5DC86"));  pDialog.setTitleText(msg);  pDialog.setCancelable(false);  pDialog.show();   }   public void ShowErrorDialog(String msg){  pDialog.changeAlertType(SweetAlertDialog.ERROR\_TYPE);  pDialog.setTitleText("Oops...");  pDialog.setContentText(msg);  pDialog.show();  }  public void ShowSuccessDialog(String mood){  pDialog.changeAlertType(SweetAlertDialog.SUCCESS\_TYPE);  pDialog.setTitleText("Success");  pDialog.setContentText(mood);  } |
| 布局 |
| <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  android:layout\_width="fill\_parent"  android:layout\_height="match\_parent"  android:background="@color/bk">   <include layout="@layout/app\_base\_toolbar" />   <android.support.v7.widget.AppCompatImageView  android:id="@+id/img\_ccx"  android:layout\_width="match\_parent"  android:layout\_height="300dp"  android:layout\_alignParentStart="true"  android:layout\_centerInParent="true"  android:layout\_below="@layout/app\_base\_toolbar"  android:src="@drawable/uma" />    <Button  style="@style/Widget.AppCompat.Button.Borderless.Colored"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignBaseline="@+id/emoji\_result"  android:clickable="false"  android:layout\_alignLeft="@+id/btn\_get\_photo"  android:text="识别结果：" />   <TextView  android:id="@+id/emoji\_result"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentBottom="true"  android:layout\_centerHorizontal="true"  android:layout\_marginBottom="145dp"  android:text="暂无"  android:textSize="16sp" />  <Button  android:id="@+id/btn\_grey\_photo"  style="@style/Widget.AppCompat.Button.Colored"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentEnd="true"  android:layout\_alignParentRight="true"  android:layout\_alignTop="@+id/btn\_get\_photo\_from\_album"  android:layout\_marginEnd="19dp"  android:layout\_marginRight="19dp"  android:text="识别表情" />   <Button  android:id="@+id/btn\_get\_photo\_from\_album"  style="@style/Widget.AppCompat.Button.Colored"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentBottom="true"  android:layout\_centerHorizontal="true"  android:layout\_marginBottom="67dp"  android:text="从相册中选图片" />   <Button  android:id="@+id/btn\_get\_photo"  style="@style/Widget.AppCompat.Button.Colored"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_alignParentStart="true"  android:layout\_alignTop="@+id/btn\_grey\_photo"  android:layout\_marginStart="27dp"  android:text="拍摄图片" />  <TextView  android:id="@+id/result"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_below="@id/gv\_about"  android:textColor="@color/font\_black\_2"  android:text="@string/app\_name"  android:layout\_marginTop="15dp"  android:textSize="@dimen/font\_size\_big"  android:layout\_centerHorizontal="true"  android:visibility="gone"/>   <LinearLayout  android:id="@+id/tv\_version\_right"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:layout\_alignParentBottom="true"  android:gravity="center\_horizontal"  android:orientation="vertical"  android:padding="10dp"  android:visibility="gone"  >   </LinearLayout>  </RelativeLayout> |

* + 1. 服务器端核心代码

|  |
| --- |
| 面部识别 |
| # 识别人脸，调整图像大小  def format\_image(image):  # 将图片转化成灰度图片  if len(image.shape) > 2 and image.shape[2] == 3:  image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  faces = cascade\_classifier.detectMultiScale(  image,  scaleFactor=1.3,  minNeighbors=5  )  # 没有面部  if not len(faces) > 0:  return None, None  # 识别最大的面部  max\_are\_face = faces[0]  for face in faces:  if face[2] \* face[3] > max\_are\_face[2] \* max\_are\_face[3]:  max\_are\_face = face  # 将面部数据保存为image  face\_coor = max\_are\_face  image = image[face\_coor[1]:(face\_coor[1] + face\_coor[2]), face\_coor[0]:(face\_coor[0] + face\_coor[3])]  # 重构image  try:  image = cv2.resize(image, (48, 48), interpolation=cv2.INTER\_CUBIC)  except Exception:  return None, None  return image, face\_coor  # 检测人脸 返回最大面部的坐标  def face\_dect(image):  # 灰度处理  if len(image.shape) > 2 and image.shape[2] == 3:  image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # 重新储存  faces = cascade\_classifier.detectMultiScale( # 识别的面部信息  image,  scaleFactor=1.3,  minNeighbors=5  )  # 没有人脸  if not len(faces) > 0:  return None  # 找到最大的面部  max\_face = faces[0]  for face in faces:  if face[2] \* face[3] > max\_face[2] \* max\_face[3]:  max\_face = face # 最大面部  face\_image = image[max\_face[1]:(max\_face[1] + max\_face[2]), max\_face[0]:(max\_face[0] + max\_face[3])] # 面部截取的图片  try: # 调整图片大小  image = cv2.resize(face\_image, (48, 48), interpolation=cv2.INTER\_CUBIC) / 255.  except Exception:  return None  return face\_image  def demo(modelPath, showBox = False, image\_path="p3.jpg"):  # 构建模型  face\_x = tf.placeholder(tf.float32, [None, 2304])  y\_conv = deepnn(face\_x)  probs = tf.nn.softmax(y\_conv)  # 存储器  saver = tf.train.Saver()  ckpt = tf.train.get\_checkpoint\_state(modelPath)  sess = tf.Session()  # 加载模型  if ckpt and ckpt.model\_checkpoint\_path:  saver.restore(sess, ckpt.model\_checkpoint\_path)  print('模型加载成功！')  # 加载emoji  feelings\_faces = []  for index, emotion in enumerate(EMOTIONS):  feelings\_faces.append(cv2.imread('./data/emojis/' + emotion + '.png', -1))  emoji\_face = []  result = None  img = load\_image\_skimage(image\_path) # 加载路径  detected\_face, face\_coor = format\_image(img) #识别的人  # 如果存在人脸，就进行表情的识别  if detected\_face is not None:  tensor = image\_to\_tensor(detected\_face)  #识别人脸的情绪，并计算情绪分类的概率  result = sess.run(probs, feed\_dict={face\_x: tensor})  face\_id = 0  if result is not None:  face\_id = np.argmax(result[0]) # 获取面部情绪信息  return face\_emoji(face\_id)  else:  return 0 |
| Model |
| def deepnn(x):  x\_image = tf.reshape(x, [-1, 48, 48, 1])  # conv1  W\_conv1 = weight\_variables([5, 5, 1, 64])  b\_conv1 = bias\_variable([64])  h\_conv1 = tf.nn.relu(conv2d(x\_image, W\_conv1) + b\_conv1)  # pool1  h\_pool1 = maxpool(h\_conv1)  # norm1  norm1 = tf.nn.lrn(h\_pool1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)  # conv2  W\_conv2 = weight\_variables([3, 3, 64, 64])  b\_conv2 = bias\_variable([64])  h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)  norm2 = tf.nn.lrn(h\_conv2, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)  h\_pool2 = maxpool(norm2)  # Fully connected layer  W\_fc1 = weight\_variables([12 \* 12 \* 64, 384])  b\_fc1 = bias\_variable([384])  h\_conv3\_flat = tf.reshape(h\_pool2, [-1, 12 \* 12 \* 64])  h\_fc1 = tf.nn.relu(tf.matmul(h\_conv3\_flat, W\_fc1) + b\_fc1)  # Fully connected layer  W\_fc2 = weight\_variables([384, 192])  b\_fc2 = bias\_variable([192])  h\_fc2 = tf.matmul(h\_fc1, W\_fc2) + b\_fc2  # linear 7类  W\_fc3 = weight\_variables([192, 7])  b\_fc3 = bias\_variable([7])  y\_conv = tf.add(tf.matmul(h\_fc2, W\_fc3), b\_fc3)  return y\_conv  def conv2d(x, W):  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')  def maxpool(x):  return tf.nn.max\_pool(x, ksize=[1, 3, 3, 1],  strides=[1, 2, 2, 1], padding='SAME')  def weight\_variables(shape):  initial = tf.truncated\_normal(shape, stddev=0.1)  return tf.Variable(initial)  def bias\_variable(shape):  initial = tf.constant(0.1, shape=shape)  return tf.Variable(initial)  def train\_model(train\_data):  fer2013 = input\_data(train\_data)  max\_train\_steps = 30001  x = tf.placeholder(tf.float32, [None, 2304])  y\_ = tf.placeholder(tf.float32, [None, 7])  y\_conv = deepnn(x)  cross\_entropy = tf.reduce\_mean(  tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y\_, logits=y\_conv))  train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)  correct\_prediction = tf.equal(tf.argmax(y\_conv, 1), tf.argmax(y\_, 1))  accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, tf.float32))  with tf.Session() as sess:  saver = tf.train.Saver()  sess.run(tf.global\_variables\_initializer())  for step in range(max\_train\_steps):  batch = fer2013.train.next\_batch(50)  if step % 100 == 0:  train\_accuracy = accuracy.eval(feed\_dict={  x: batch[0], y\_: batch[1]})  print('step %d, training accuracy %g' % (step, train\_accuracy))  train\_step.run(feed\_dict={x: batch[0], y\_: batch[1]})  if step + 1 == max\_train\_steps:  saver.save(sess, './models/emotion\_model', global\_step=step + 1)  if step % 1000 == 0:  print('\*Test accuracy %g' % accuracy.eval(feed\_dict={  x: fer2013.validation.images, y\_: fer2013.validation.labels}))  def image\_to\_tensor(image):  tensor = np.asarray(image).reshape(-1, 2304) \* 1 / 255.0  return tensor |

* 1. 心理咨询模块核心代码

|  |
| --- |
| 心理咨询 |
| // 判断今日状态  public boolean JudgeIfTodayRecord(){  DataBase dbHelper = new DataBase(this, "Chatbot.db", null, 1); // 数据库  SQLiteDatabase db = dbHelper.getReadableDatabase();// 打开数据库  String sql = "select \* from MoodRecord where date = '"+getMyDate()+"'";  Cursor cursor = db.rawQuery(sql, null); // 执行SQL语句  if(cursor.getCount()==0) {db.close();cursor.close();return false;} // 为负面  else {db.close();cursor.close();return true;} }  // 判断今日心情是否记录 public void FirstMood(){  if(JudgeIfTodayRecord()) return; // 已经记录  HideInput(); //隐藏用户输入操作界面  setLeft("哈喽~又是新的一天，今天的心情怎么样呢？\n(\*＾-＾\*)");  ShowMoodBtn(); // 显示心情选择按钮 }  // 心情记录 public void MoodRecord(){  DataBase dbHelper = new DataBase(this, "Chatbot.db", null, 1);  SQLiteDatabase db = dbHelper.getWritableDatabase();// 打开数据库  ContentValues values = new ContentValues(); // 存储值  values.put("date", getMyDate()); // 获取日期，存在date中  values.put("mood", mood); // 获取心情，存在mood中  db.insert("MoodRecord", null, values); // 执行插入操作  Toast.makeText(this, "添加成功", Toast.LENGTH\_LONG).show();  db.close(); }  // 具体咨询内容操作，state 为跳转的状态点 public void Help(int state){   switch (now\_state){ // 选择状态点  case 0: break;  case 1:  setLeft("告诉我你的烦恼吧，也许我可以帮到你。");  break;  case 2:  setLeft("想要尝试缓解心理压力吗？");  HideInput();  ShowYesOrNoBtn("尝试一下","下次吧");  break;  case 3: setLeft(seq.MoodTalk[4]);HideInput();ShowProblemBtn();break;  case 4: setLeft(seq.MoodTalk[5]);ShowNothingBtn("是的");break;  case 5: setLeft(seq.MoodTalk[6]);ShowYesOrNoBtn("改善情绪","尝试放松");break;  case 6: setLeft(seq.MoodTalk[7]);ShowNothingBtn("好的");break;  case 7: setLeft(seq.MoodTalk[8]);setLeft(seq.MoodTalk[9]);ShowNothingBtn("好的");break;  case 8: setLeft(seq.MoodTalk[10]);setLeft(seq.MoodTalk[11]);setLeft(seq.MoodTalk[12]);ShowNothingBtn("放松中");break;  case 9: setLeft(seq.MoodTalk[13]);ShowInput();now\_state=0;break; //Ask=0;  case 10: setLeft(seq.MoodTalk[14]);setLeft(seq.MoodTalk[15]);setLeft(seq.MoodTalk[16]);ShowNothingBtn("什么事情？");break;  case 11: setLeft(seq.MoodTalk[17]);setLeft(seq.MoodTalk[18]);ShowNothingBtn("是的");break;  case 12: setLeft(seq.MoodTalk[19]);setLeft(seq.MoodTalk[20]);ShowNothingBtn("明白");break;  case 13: setLeft(seq.MoodTalk[21]);setLeft(seq.MoodTalk[22]);ShowNothingBtn("后来呢");break;  case 14: setLeft(seq.MoodTalk[23]);setLeft(seq.MoodTalk[24]);setLeft(seq.MoodTalk[25]);ShowNothingBtn("但愿如此");break;  case 15: setLeft(seq.MoodTalk[26]);ShowNothingBtn("谢谢");break;  case 16: setLeft(seq.MoodTalk[27]);ShowInput();now\_state=0;break;  case 17: setLeft(seq.MoodTalk[28]);setLeft(seq.MoodTalk[29]);setLeft(seq.MoodTalk[30]);ShowNothingBtn("我觉得是这样");break;  case 18: setLeft(seq.MoodTalk[31]);ShowYesOrNoBtn("重塑思维","认清事实");break;   case 19: setLeft(seq.MoodTalk[32]);ShowNothingBtn("是的");break;  case 20: setLeft(seq.MoodTalk[33]); setLeft(seq.MoodTalk[34]);ShowNothingBtn("另一方面呢？");break;  case 21: setLeft(seq.MoodTalk[35]);ShowNothingBtn("明白");break;  case 22: setLeft(seq.MoodTalk[36]);ShowNothingBtn("寻找问题的关键");break;  case 23: setLeft(seq.MoodTalk[37]);ShowYesOrNoBtn("好消息","坏消息");news=0;break;  case 24: setLeft(seq.MoodTalk[38]);if(news==0) {ShowNothingBtn("那坏消息呢？");news++;}  else{ShowNothingBtn("该怎么做呢？");news++;} break;  case 25: setLeft(seq.MoodTalk[39]); if(news==0) {ShowNothingBtn("那好消息呢？");news++;}  else {ShowNothingBtn("该怎么做呢？");news++;} break;  case 26: setLeft(seq.MoodTalk[40]);ShowNothingBtn("开始吧");news=0;break;  case 27: setLeft(seq.MoodTalk[41]); setLeft(seq.MoodTalk[42]); HideBtn(); ShowInput();break;  case 28: setLeft(seq.MoodTalk[43]); break;  case 29: setLeft(seq.MoodTalk[44]); setLeft(seq.MoodTalk[45]); break;  case 30: setLeft(seq.MoodTalk[46]); break;  case 31: setLeft(seq.MoodTalk[47]);HideInput();ShowYesOrNoBtn("有","没有");break;  case 32: setLeft(seq.MoodTalk[48]);ShowYesOrNoBtn("有","没有");break;  case 33: setLeft(seq.MoodTalk[49]);ShowYesOrNoBtn("更糟糕","更好");break;  case 34: setLeft(seq.MoodTalk[50]); setLeft(seq.MoodTalk[51]); setLeft(seq.MoodTalk[52]);ShowNothingBtn("我懂了");break;  case 35: setLeft(seq.MoodTalk[53]); HideBtn(); ShowInput();break;  case 36: setLeft(seq.MoodTalk[54]); break;  case 37: setLeft(seq.MoodTalk[55]); break;  case 38: setLeft(seq.MoodTalk[56]); break;  case 39: setLeft(seq.MoodTalk[57]); setLeft("一开始你说：“"+A+"”，现在你认为“" +B+"，"+C+"，"+D+"，"+E+"。"); setLeft(seq.MoodTalk[59]); setLeft(seq.MoodTalk[60]);  ShowInput();now\_state=0;break;   case 40: setLeft(seq.MoodTalk[61]); setLeft(seq.MoodTalk[62]); HideInput();ShowYesOrNoBtn("入眠困难","常有噩梦"); break;  case 41: setLeft(seq.MoodTalk[63]); setLeft(seq.MoodTalk[64]); ShowNothingBtn("是的");break;  case 42: setLeft(seq.MoodTalk[65]); setLeft(seq.MoodTalk[66]); setLeft(seq.MoodTalk[67]); setLeft(seq.MoodTalk[68]); setLeft(seq.MoodTalk[69]);  setLeft(seq.MoodTalk[70]); setLeft(seq.MoodTalk[71]); break;  case 43: setLeft(seq.MoodTalk[72]); setLeft(seq.MoodTalk[73]);ShowNothingBtn("真的吗？");break;  case 44: setLeft(seq.MoodTalk[74]); setLeft(seq.MoodTalk[75]);ShowNothingBtn("什么方法？");break;  case 45: setLeft(seq.MoodTalk[76]); setLeft(seq.MoodTalk[77]);ShowInput();now\_state=0;break;   case 46: setLeft(seq.MoodTalk[78]); ShowYesOrNoBtn("尝试缓解","认清事实");break;   case 47: setLeft(seq.CBT[0]);setLeft(seq.CBT[1]);ShowInput(); break;  case 48: setLeft(seq.CBT[2]);HideInput();ShowNothingBtn("什么是认知扭曲？");break;  case 49: setLeft(seq.CBT[3]);setLeft(seq.CBT[4]);setLeft(seq.CBT[5]); ShowYesOrNoBtn("是的","不存在");break;  case 50: setLeft(seq.CBT[6]);ShowNothingBtn("没错");break;  case 51: setLeft(seq.CBT[7]);ShowNothingBtn("非黑即白？");break;  case 52: setLeft(seq.CBT[8]);setLeft(seq.CBT[9]); ShowYesOrNoBtn("是的","不存在");break;  case 53: setLeft(seq.CBT[10]); now\_state=53;  case 54: setLeft(seq.CBT[11]); now\_state=54;ShowYesOrNoBtn("存在","不存在");break;  case 55: setLeft(seq.CBT[12]); Toast.makeText(this,"!!",Toast.LENGTH\_LONG).show();ShowNothingBtn("有趣");break;  case 56: setLeft(seq.CBT[13]); ShowYesOrNoBtn("是的","并没有");break;  case 57: setLeft(seq.CBT[14]); setLeft(seq.CBT[15]); ShowNothingBtn("的确");break;  case 58: setLeft(seq.CBT[16]); ShowNothingBtn("就是这样，没错");break;  case 59: setLeft(seq.CBT[17]); ShowNothingBtn("谢谢~");break;  case 60: setLeft(seq.CBT[18]); setLeft("你原先的想法是："+A); ShowInput(); break;  case 61: setLeft(seq.CBT[19]); HideInput();ShowYesOrNoBtn("好多了","不知道该怎么办");break;  case 62: setLeft(seq.CBT[20]); ShowNothingBtn("比如说？");break;  case 63: setLeft(seq.CBT[21]); setLeft(seq.CBT[22]);ShowNothingBtn("大概懂了"); break;  case 64: setLeft(seq.CBT[23]); now\_state=0;ShowInput();HideBtn();break;    case 65: setLeft(seq.CTF[0]); setLeft(seq.CTF[1]); setLeft(seq.CTF[2]); setLeft(seq.CTF[3]); ShowNothingBtn("我懂了");break;  case 66: setLeft(seq.CTF[4]); setLeft(seq.CTF[5]); ShowNothingBtn("现在就开始吧");break;  case 67: setLeft(seq.CTF[6]); setLeft(seq.CTF[7]); ShowNothingBtn("真是糟糕");break;  case 68: setLeft(seq.CTF[8]); setLeft(seq.CTF[9]);setLeft(seq.CTF[10]); ShowNothingBtn("事实却可能不是这样的");break;  case 69: setLeft(seq.CTF[11]); ShowInput(); break;  case 70: setLeft(seq.CTF[12]);setLeft(seq.CTF[13]);setLeft(seq.CTF[14]);setLeft(seq.CTF[15]);setLeft(seq.CTF[16]);HideInput();ShowNothingBtn("哇");break;  case 71: setLeft(seq.CTF[17]); ShowNothingBtn("没错");break;  case 72: setLeft(seq.CTF[18]);setLeft(seq.CTF[19]);setLeft(seq.CTF[20]); ShowInput(); break;  case 73: setLeft(seq.CTF[21]); break;  case 74: setLeft("接下来我们要列出所有"+B+"发展中存在的潜在可能性。"); break;  case 75: setLeft("现在我希望你能更客观的描述这个造成你"+A+"的事情……意思是，从你眼睛、鼻子、耳朵中直接得到的信息。");setLeft(seq.CTF[24]); break;  case 76: setLeft(seq.CTF[25]);setLeft(seq.CTF[26]); now\_state=0;break;  } } |
| 布局 |
| <LinearLayout  xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:background="@color/bk"  android:orientation="vertical">   <include layout="@layout/app\_base\_toolbar"/>   <RelativeLayout   android:layout\_width="match\_parent"  android:layout\_height="match\_parent">   <ListView // 对话布局  android:id="@+id/lv\_message2"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:layout\_above="@+id/divider2"  android:divider="#00000000"  android:dividerHeight="0px"  android:transcriptMode="normal"  />   <View // 分割线  android:id="@+id/divider2"  android:layout\_width="match\_parent"  android:layout\_height="1dp"  android:layout\_above="@+id/btn\_layout"  android:background="@color/font\_black\_5"/>   <LinearLayout //各种按钮  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_layout"  android:orientation="vertical"  android:layout\_above="@+id/rl\_msg2"  android:gravity="center\_vertical"  >  <Button   android:id="@+id/btn\_yes"  android:visibility="gone"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:background="#FFFFFF"  android:text="是的"/>  <Button  android:id="@+id/btn\_no"  android:visibility="gone"  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:background="#FFFFFF"  android:text="不是" />   <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_nothing"  android:background="#FFFFFF"  android:visibility="gone"  />   <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_anxious"  android:background="#FFFFFF"  android:visibility="gone"  android:text="焦虑"  />   <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_regret"  android:background="#FFFFFF"  android:visibility="gone"  android:text="后悔"  />   <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_angry"  android:background="#FFFFFF"  android:visibility="gone"  android:text="生气"  />   <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_ship"  android:background="#FFFFFF"  android:visibility="gone"  android:text="人际关系"  />   <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_lonely"  android:background="#FFFFFF"  android:visibility="gone"  android:text="孤独"  />   <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_sleep"  android:background="#FFFFFF"  android:visibility="gone"  android:text="睡眠问题"  />   <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_fine"  android:background="#FFFFFF"  android:visibility="gone"  android:text="挺好的"  />  <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_1"  android:background="#FFFFFF"  android:visibility="gone"  android:text="开心"  />  <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_2"  android:background="#FFFFFF"  android:visibility="gone"  android:text="平静"  />  <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_3"  android:background="#FFFFFF"  android:visibility="gone"  android:text="难过"  />  <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_4"  android:background="#FFFFFF"  android:visibility="gone"  android:text="抑郁"  />  <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_5"  android:background="#FFFFFF"  android:visibility="gone"  android:text="焦虑"  />  <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_6"  android:background="#FFFFFF"  android:visibility="gone"  android:text="生气"  />  <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_7"  android:background="#FFFFFF"  android:visibility="gone"  android:text="害怕"  />  <Button  android:layout\_width="match\_parent"  android:layout\_height="wrap\_content"  android:id="@+id/btn\_8"  android:background="#FFFFFF"  android:visibility="gone"  android:text="疲惫"  />  </LinearLayout>   <RelativeLayout  android:id="@+id/rl\_msg2"  android:layout\_width="fill\_parent"  android:layout\_height="50dp"  android:layout\_alignParentBottom="true"  android:gravity="center\_vertical">   <ImageView  android:id="@+id/iv\_send\_msg2"  android:layout\_width="30dp"  android:layout\_height="30dp"  android:layout\_alignParentRight="true"  android:layout\_centerVertical="true"  android:layout\_marginRight="10dp"  android:src="@drawable/selector\_send"/>   <LinearLayout  android:id="@+id/ll\_msg2"  android:layout\_width="match\_parent"  android:layout\_height="32dp"  android:layout\_centerVertical="true"  android:layout\_marginLeft="10dp"  android:layout\_marginRight="10dp"  android:layout\_toLeftOf="@+id/iv\_send\_msg2"  android:background="@drawable/actionsheet\_single\_normal"  android:orientation="vertical">   <EditText  android:id="@+id/et\_msg2"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:layout\_gravity="center\_vertical"  android:background="@color/white"  android:hint="输入您的问题"  android:imeOptions="actionSend|flagNoEnterAction"  android:inputType="textCapSentences|textAutoCorrect|textMultiLine|textShortMessage"  android:maxLength="100"  android:maxLines="4"  android:minHeight="32dp"  android:paddingLeft="10dp"  android:paddingRight="10dp"  android:textColor="@color/font\_black\_2"  android:textColorHint="@color/font\_black\_5"  android:textSize="14sp"/>  </LinearLayout>  </RelativeLayout>  </RelativeLayout> </LinearLayout> |

* 1. 心情记录模块核心代码

|  |
| --- |
| 心情记录显示 |
| private void getRecord(){ // 获取历史心情  DataBase dbHelper = new DataBase(this, "Chatbot.db", null, 1);  SQLiteDatabase db = dbHelper.getReadableDatabase();// 打开数据库  String sql = "select \* from MoodRecord where mood = '开心'";  Cursor cursor = db.rawQuery(sql, null); //执行SQL语句  HAPPY = cursor.getCount(); // 开心次数  sql = "select \* from MoodRecord where mood = '平静'";  cursor = db.rawQuery(sql, null);   NOTHING = cursor.getCount(); // 平静次数  sql = "select \* from MoodRecord where mood = '难过'";  cursor = db.rawQuery(sql, null);  SAD = cursor.getCount(); // 难过次数  sql = "select \* from MoodRecord where mood = '生气'";  cursor = db.rawQuery(sql, null);  ANGERY = cursor.getCount(); // 生气次数  sql = "select \* from MoodRecord where mood = '焦虑'";  cursor = db.rawQuery(sql, null);  ANXIOUS = cursor.getCount();// 焦虑次数  sql = "select \* from MoodRecord where mood = '生病'";  cursor = db.rawQuery(sql, null);  SILK = cursor.getCount(); // 生病次数  sql = "select \* from MoodRecord where mood = '抑郁'";  cursor = db.rawQuery(sql, null);  DEPRESSION = cursor.getCount(); // 抑郁次数  sql = "select \* from MoodRecord where mood = '疲惫'";  cursor = db.rawQuery(sql, null);  TIRED = cursor.getCount(); // 疲惫次数  cursor.close();  db.close(); }  private void initPie(){  //饼状图  mPieChart = (PieChart) findViewById(R.id.mPieChart);  mPieChart.setUsePercentValues(true);  mPieChart.getDescription().setEnabled(false);  mPieChart.setExtraOffsets(5, 10, 5, 5);   mPieChart.setDragDecelerationFrictionCoef(0.95f);  //设置中间文件  mPieChart.setCenterText("今日："+GetMood());   mPieChart.setDrawHoleEnabled(true);  mPieChart.setHoleColor(Color.WHITE);   mPieChart.setTransparentCircleColor(Color.WHITE);  mPieChart.setTransparentCircleAlpha(110);   mPieChart.setHoleRadius(58f);  mPieChart.setTransparentCircleRadius(61f);   mPieChart.setDrawCenterText(true);   mPieChart.setRotationAngle(0);  // 触摸旋转  mPieChart.setRotationEnabled(true);  mPieChart.setHighlightPerTapEnabled(true);   getRecord();  //变化监听  //模拟数据  ArrayList<PieEntry> entries = new ArrayList<PieEntry>();  if(HAPPY!=0) entries.add(new PieEntry(HAPPY, "开心"));  if(NOTHING!=0) entries.add(new PieEntry(NOTHING, "平静"));  if(SAD!=0) entries.add(new PieEntry(SAD, "难过"));  if(DEPRESSION!=0) entries.add(new PieEntry(DEPRESSION, "抑郁"));  if(ANGERY!=0) entries.add(new PieEntry(ANGERY, "生气")); // 惊讶，恐惧，恶心  if(ANXIOUS!=0) entries.add(new PieEntry(ANXIOUS, "焦虑"));  if(SILK!=0) entries.add(new PieEntry(SILK, "生病"));  if(TIRED!=0) entries.add(new PieEntry(TIRED, "疲惫"));   //设置数据  setData(entries);  mPieChart.animateY(1400, Easing.EasingOption.EaseInOutQuad);   Legend l = mPieChart.getLegend();  l.setVerticalAlignment(Legend.LegendVerticalAlignment.TOP);  l.setHorizontalAlignment(Legend.LegendHorizontalAlignment.RIGHT);  l.setOrientation(Legend.LegendOrientation.VERTICAL);  l.setDrawInside(false);  l.setXEntrySpace(7f);  l.setYEntrySpace(0f);  l.setYOffset(0f);   // 输入标签样式  mPieChart.setEntryLabelColor(Color.WHITE);  mPieChart.setEntryLabelTextSize(12f);  //设置数据  for (IDataSet<?> set : mPieChart.getData().getDataSets()){  set.setDrawValues(!set.isDrawValuesEnabled());  }  mPieChart.invalidate();  mPieChart.animateXY(1400, 1400); } |
| 布局 |
| <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  android:layout\_width="fill\_parent"  android:layout\_height="match\_parent"  android:background="@color/bk">   <include layout="@layout/app\_base\_toolbar" />  <LinearLayout // 线性布局  android:layout\_below="@+id/toolbar"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:layout\_centerInParent="true"  android:orientation="vertical">   <com.github.mikephil.charting.charts.PieChart // 饼状图  android:id="@+id/mPieChart"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  android:layout\_weight="1"/>     </LinearLayout> </RelativeLayout> |