详细设计说明书

1 引言

1.1编写目的

本说明是基于机器学习的自动音乐生成软件——Composer软件产品的详细设计和实现说明，本文档记录了系统整体实现上技术层面上的考虑，并且以需求说明作为依据，同时该文档将作为产品实现、特性要求和控制的依据。

软件开发小组的每一位参与开发成员应该阅读本说明，以清楚产品在技术方面的要求和实现策略，本手册将进行技术评审和技术的可行性检查，同时为下一步的测试报告提供框架。此外，软件开发小组的指导老师也可以通过阅读本文档，了解开发小组在软件开发过程中的详细设计，从而给出宝贵的意见和建议。

1.2背景

说明：

1. 待开发软件系统的名称：基于机器学习的自动音乐生成播放器——Composer
2. 此项目的任务提出者：西电软件工程课程组、西电软件开发小组

开发者：西电软件开发小组

将运行该软件的计算站（中心）：用户本地运行、服务器运行后端

1.3定义

A. LSTM （Long Short-Term Memory 长短期记忆网络）

B. UI （User Interface 用户界面）

1.4参考资料

相关的文件：

A. 软件介绍《基于机器学习的自动音乐生成播放器——Composer》

B. 《基于机器学习的自动音乐生成播放器软件需求分析报告》

参考资料：

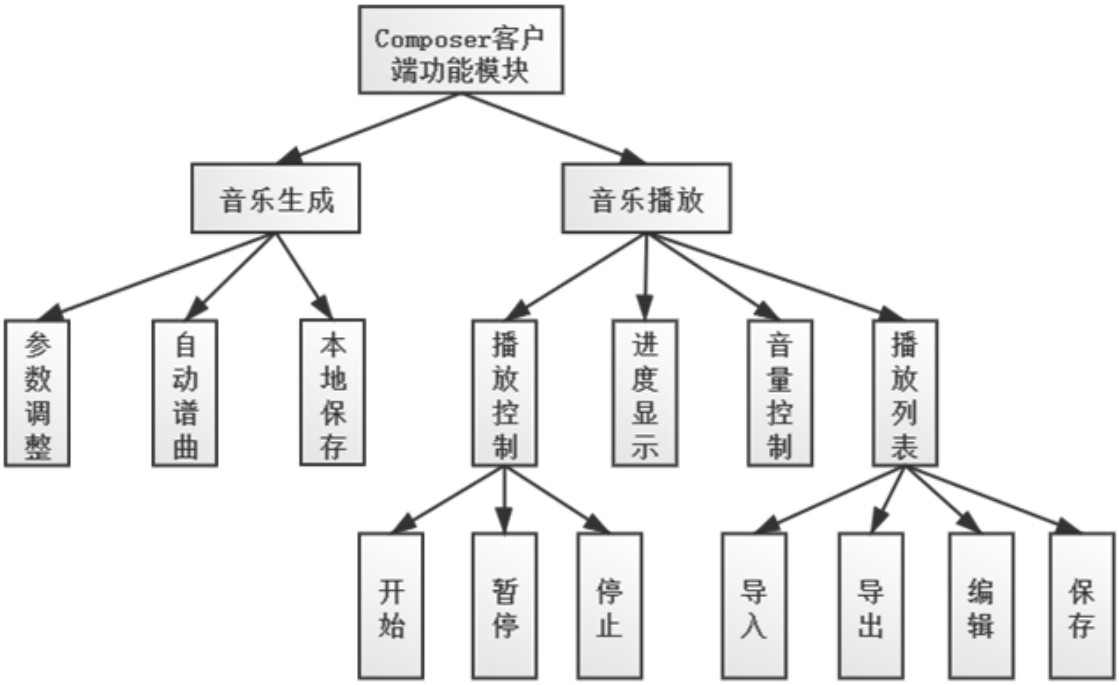
A. 《2020级软件工程版 课程实践题目及介绍》

B. 国家标准《概要设计说明书(GB8567-88)》

C. 《软件工程导论（第六版）》清华大学出版社

C. 《LSTM（长短期记忆网络）介绍与实现》

2 程序系统的结构



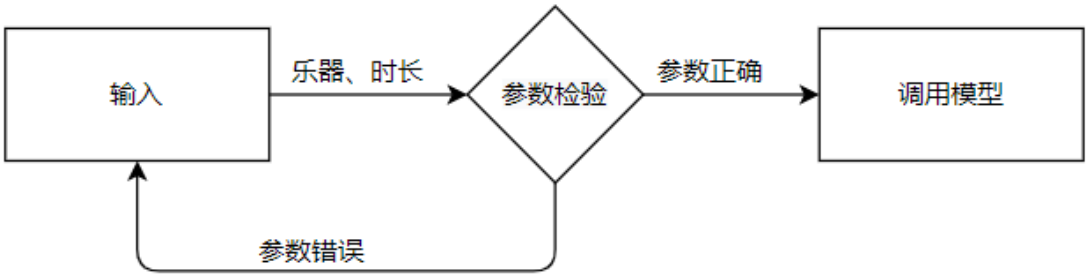
系统由2大模块组成，7小模块组成：

序号	编号	名称
1	01	音乐生成模块
2	011	参数调整模块
3	012	自动谱曲模块
4	013	本地保存模块
5	02	音乐播放模块
6	021	播放控制模块
7	022	进度显示模块
8	023	音量控制模块
9	024	播放列表模块

3 程序描述

3.1 011 参数调整模块

3.1.1 功能流程图



3.1.2 功能描述

（1）功能类型：参数调整

（2）功能描述：检验用户输入参数的正确性

（3）前提业务：无

（4）后继业务：012（自动谱曲模块）

（5）功能约束：无

3.1.3 界面设计

（1）基础信息处理

动作说明：

动作编号	动作名称	动作描述
A01	调整时长	拖动时长按钮，选择用户希望生成音乐的大致时长
A02	选择乐器	通过乐器选择按钮，选择生成音乐的演奏乐器

（2）数据要求

1.功能类型：参数输入

2.数据描述

字段名称	长度	录入方式	是否非空项	数据检验	默认显示
时长	4	Scale	Y	N	
乐器	10	ListBox	Y	N	

3.1.4 参数调整的内部逻辑

1.参数输入

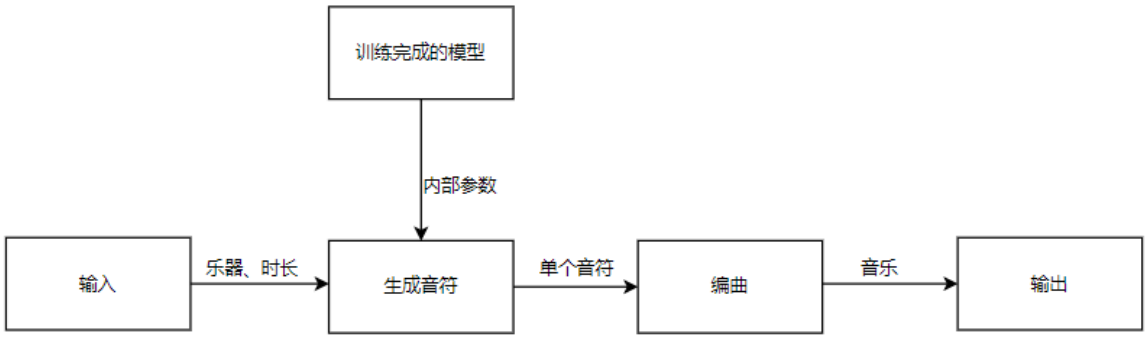
```
# 第4步，在图形界面上创建一个标签label用以显示并放置
var1 = StringVar()  # 创建变量，用var1用来接收鼠标点击具体选项的内容
l = Label(root, bg='white', fg='black',font=('Arial', 12), width=10,
textvariable=var1)
l.grid(row =2,column =2)

# 第6步，创建一个方法用于按钮的点击事件
def print_selection():
    value = lb.get(lb.curselection())    # 获取当前选中的文本
    var1.set(value)  # 为label设置值
    print(var1.get())

# 第5步，创建一个按钮并放置，点击按钮调用print_selection函数
b1 = Button(root, text='selection instrument', width=18, height=2,
command=print_selection)
b1.grid(row =3,column =2)

# 第7步，创建Listbox并为其添加内容
var2 = StringVar()
var2.set(('Piano','Flute','Bass','Guitar','Saxophone','Violin')) # 为变量var2设置值
# 创建Listbox
lb = Listbox(root, listvariable=var2)  #将var2的值赋给Listbox
# 创建一个list并将值循环添加到Listbox控件中
lb.grid(row =4,column =2)

var3 = StringVar()
l2 = Label(root, bg='white', fg='black', width=20)
l2.grid(row =8,column =2)

def selection(v):
    var3.set(float(v))
    l2.config(text='duration: ' + v)
s = Scale(root, label='select music duration', from_=0, to=10,
orient=HORIZONTAL, length=200, showvalue=0,tickinterval=2, resolution=0.01,
command=selection)
s.grid(row =9,column =2)
```

2.传递参数

```
def generate_music():
    print(str(var3.get())+str(type(var3.get())))
    generate(instrument=var1.get(), filename=var1.get()+ '-'+time.strftime("%Y-
%m-%d-%H-%M-%S", time.localtime()), duration = var3.get())
```

3.2 012 自动谱曲模块

3.2.1 功能流程图



3.2.2 功能描述

（1）功能类型：自动谱曲

（2）功能描述：根据已训练完成的模型和用户生成的参数生成音乐

（3）前提业务：011（参数调整模块）

（4）后继业务：013（本地保存模块）

（5）功能约束：权限约束

（6）约束描述：只能由软件内部调用

3.2.3 界面设计

本模块不提供用户交互界面

3.2.4 自动谱曲的内部逻辑

1.读取已训练模型参数

```
def get_data(filename):
    """从文件中获取音符

    :param filename: [文件名]
    :type filename: [str]
    :return: [返回音符]
    :rtype: [list]
    """
    with open(filename) as f:
       all_notes = f.readlines()
    return [ note[:len(note)-1]  for note in all_notes]
    # 从保存的数据集中获得数据
all_notes = get_data("data.txt")
# 加载模型
from keras.models import load_model
model = load_model("weights-804-0.01.hdf5")
```

2.预测音符

```
def predict_next(X_predict, model):
    """通过前100个音符，预测下一个音符

    :param X_predict: [前100个音符]
    :type X_predict: [list]
    :return: [下一个音符的id]
    :rtype: [int]
    """
    prediction = model.predict(X_predict)
    index = np.argmax(prediction)
    return index

def generate_notes(X_one_hot, id_to_note, X_train,model, duration):
    """随机从X_one_hot抽取一个数据（长为100），然后进行predict，最后生成音乐
    :return: [note数组（['D5', '2.6', 'F#5', 'D3', ……]）]
    :rtype: [list]
    """
    # 随机从X_one_hot选择一个数据进行predict
    randindex = np.random.randint(0, len(X_one_hot) - 1)
    predict_input = X_one_hot[randindex]
    # music_output里面是一个数组，如['D5', '2.6', 'F#5', 'D3', 'E5', '2.6', 'G5',
'F#5']
    music_output = [id_to_note[id] for id in X_train[randindex]]
    # 产生长度为1000的音符序列
    for note_index in range(int(float(duration)*100)):
        prediction_input = np.reshape(predict_input,
(1,X_one_hot.shape[1],X_one_hot.shape[2]))
        # 预测下一个音符id
        predict_index = predict_next(prediction_input,model)
        # 将id转换成音符
        music_note = id_to_note[predict_index]
        music_output.append(music_note)
        # X_one_hot.shape[-1] = 308
        one_hot_note = np.zeros(X_one_hot.shape[-1])
        one_hot_note[predict_index] = 1
        one_hot_note = np.reshape(one_hot_note,(1,X_one_hot.shape[-1]))
        # 重新构建LSTM的输入
        predict_input = np.concatenate((predict_input[1:],one_hot_note))
    return music_output
```

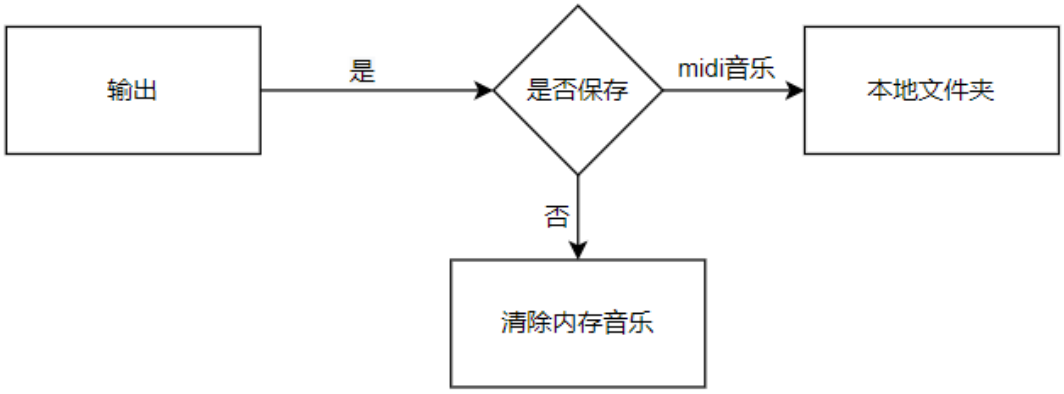
3.组成音乐

```
def generate_music(result_data, instr, filename):
    """生成mid音乐，然后进行保存

    :param result_data: [音符列表]
    :type result_data: [list]
    :param filename: [文件名]
    :type filename: [str]
    """
    result_data = [str(data) for data in result_data]
    offset = 0
    output_notes = []
    # 生成 Note（音符）或 Chord（和弦）对象
    for data in result_data:
```

```
        if ('.' in data) or data.isdigit():
            notes_in_chord = data.split('.')
            notes = []
            if instr == 'Flute':
                output_notes.append(instrument.Flute())
            elif instr == 'Piano':
                output_notes.append(instrument.Piano())
            elif instr == 'Bass':
                output_notes.append(instrument.Bass())
            elif instr == 'Guitar':
                output_notes.append(instrument.Guitar())
            elif instr == 'Saxophone':
                output_notes.append(instrument.Saxophone())
            elif instr == 'Violin':
                output_notes.append(instrument.Violin())

            for current_note in notes_in_chord:
                new_note = note.Note(int(current_note))
                #new_note.storedInstrument = instrument.Flute()
                notes.append(new_note)
            new_chord = chord.Chord(notes)
            new_chord.offset = offset
            output_notes.append(new_chord)

        else:
            if instr == 'Flute':
                output_notes.append(instrument.Flute())
            elif instr == 'Piano':
                output_notes.append(instrument.Piano())
            elif instr == 'Bass':
                output_notes.append(instrument.Bass())
            elif instr == 'Guitar':
                output_notes.append(instrument.Guitar())
            elif instr == 'Saxophone':
                output_notes.append(instrument.Saxophone())
            elif instr == 'Violin':
                output_notes.append(instrument.Violin())
            new_note = note.Note(data)
            new_note.offset = offset
            #new_note.storedInstrument = instrument.Flute()
            output_notes.append(new_note)
        offset += 1
    # 创建音乐流（Stream）
    midi_stream = stream.Stream(output_notes)
    # 写入 MIDI 文件
    midi_stream.write('midi', fp=filename+'.mid')
```

3.3 013 本地保存模块

3.3.1 功能流程图



3.3.2 功能描述

（1）功能类型：本地保存

（2）功能描述：将生成的音乐选择保存到本地

（3）前提业务：011（参数调整模块）、012（自动谱曲模块）

（4）后继业务：无

（5）功能约束：权限约束

（6）约束描述：只有获取版权的用户可以保存

3.3.3 界面设计

（1）基础信息处理

动作说明：

动作编号	动作名称	动作描述
A01	保存	点击按钮，将音乐保存到用户本地文件夹
A02	取消	点击按钮，退出此模块

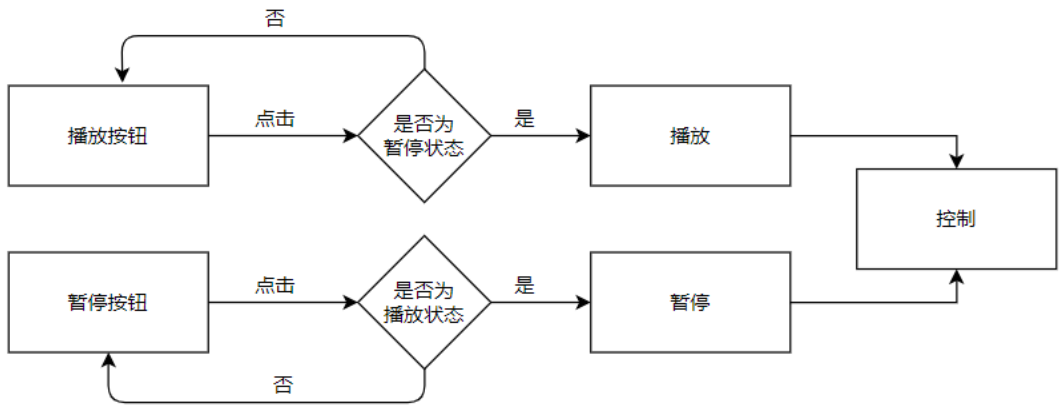
3.3.4 本地保存的内部逻辑

```
for i in range(len(on_off1)):
    if on_off1[i]:
        outtrack1.append(mido.Message('note_on', note=note1_on.pop(0),
velocity=velocity1_on.pop(0), time=time1_on.pop(0)))
    else:
        outtrack1.append(mido.Message('note_off', note=note1_off.pop(0),
velocity=velocity1_off.pop(0), time=time1_off.pop(0)))


for i in range(len(on_off2)):
    if on_off2[i]:
        outtrack2.append(mido.Message('note_on', note=note2_on.pop(0),
velocity=velocity2_on.pop(0), time=time2_on.pop(0)))
    else:
        outtrack2.append(mido.Message('note_off', note=note2_off.pop(0),
velocity=velocity2_off.pop(0), time=time2_off.pop(0)))
```

```
outfile.save('createmusic.mid')
```

3.4 021 播放控制模块

3.4.1 功能流程图



3.4.2 功能描述

（1）功能类型：播放控制

（2）功能描述：为用户提供midi音乐的播放功能

（3）前提业务：无

（4）后继业务：022（进度显示模块）、024（播放列表模块）

（5）功能约束：无

3.4.3 界面设计

（1）基础信息处理

动作说明：

动作编号	动作名称	动作描述
A01	播放	点击按钮，用户播放当前列表首位的音乐
A02	暂停	点击按钮，用户暂停播放当前音乐

3.4.4 播放控制的内部逻辑

1.函数定义

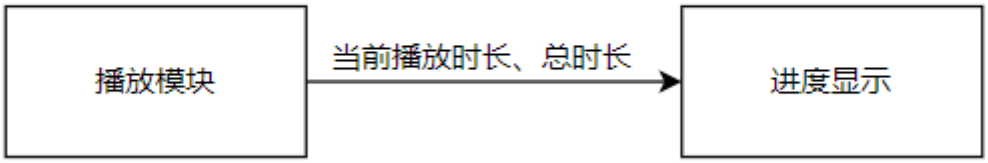
```
def play(event = None):
  #root.title("%s" % name[-1]),使用wmp.currentMedia.name更好,在per函数中
  per_thread = threading.Thread(target = per)
  per_thread.daemnon = True
  wmp.controls.play()
  per_thread.start()
def per():
  global total
  while wmp.playState !=1:
    progress_scal.set(int(wmp.controls.currentPosition))
    progress_scal.config(label = wmp.controls.currentPositionString)
    progress_scal.config(to = total,tickinterval = 50)
    time.sleep(1)
    root.title("%s" % wmp.currentMedia.name)
def stop():
  wmp.controls.stop()
def pause(event = None):
  wmp.controls.pause()
```

2.调用函数

```
ctr_lab = LabelFrame(root,text = "播放控制",height =130)
ctr_lab.grid(row =0,column =1,rowspan =12,sticky = "ns")
btn_stop = Button(ctr_lab,text ="停止",width =10,command = stop)
btn_stop.grid(row =2,column =1,pady =5)
btn_pause = Button(ctr_lab,text ="暂停",width =10,command = pause)
btn_pause.grid(row =3,column =1,pady =5)
```

3.5 022 进度显示模块

3.5.1 功能流程图



3.5.2 功能描述

（1）功能类型：进度显示

（2）功能描述：根据已训练完成的模型和用户生成的参数生成音乐

（3）前提业务：022（播放控制模块）

（4）后继业务：无

（5）功能约束：无

（6）约束描述：无

3.5.3 界面设计

3.5.4 进度显示的内部逻辑

1.定义函数

```
def openfile(index = [1]):
  global total,name

  filenames = filedialog.askopenfilenames(title = "Composer",filetypes =[("mp3文件","*.mp3"),("WMA文件","*.wma"),("WAV文件","*.wav"),("MIDI文件","*.mid")])
  print(filenames)

  if filenames:
    for i in range(len(filenames)):

      media = wmp.newMedia(filenames[i])
      wmp.currentPlaylist.appendItem(media)
      print(filenames[i][-3:])
      if filenames[i][-3:] == 'mid':
        mid = MidiFile(filenames[i])
        total = int(mid.length)
        minute = int(mid.length)//60
        sec = int(mid.length)%60
        length = int(mid.length)
      else:
          coco = eyed3.load(filenames[i])#eyed3模块读取mp3信息
          total = int(coco.info.time_secs)
          minute = int(coco.info.time_secs)//60
          sec = int(coco.info.time_secs)%60
          length = int(coco.info.time_secs)


      name = filenames[i].split("/")

      i =index[-1]
      list_name.insert(END,str(i)+"."+name[-1])
      list_name.insert(END," "*6)
      if sec >=10:
        list_name.insert(END,"0%d:%d" %(minute,sec)+ "\n")
      else:
        list_name.insert(END,"0%s:0%d" %(minute,sec)+ "\n")
      i = i +1
```

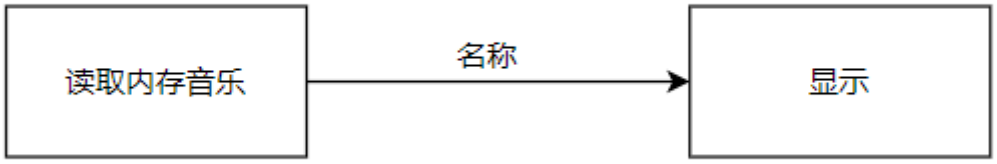
```
        index.append(i)
```

2.调用函数

```
progress_lab = LabelFrame(root,text = "播放进度")
progress_lab.grid(row =2,column =0,sticky = "we",rowspan = 2)
var_scale = DoubleVar()
progress_scal = Scale(progress_lab,orient = HORIZONTAL,showvalue = 0,length
=180,variable = var_scale)
progress_scal.bind("<Button-1>",pause)
progress_scal.bind("")
progress_scal.bind("<ButtonRelease-1>",play)
progress_scal.grid(row =3,column =0)
```

3.6 023 播放列表模块

3.6.1 功能流程图



3.6.2 功能描述

（1）功能类型：播放列表

（2）功能描述：显示当前用户正在和将要播放的音乐

（3）前提业务：021（播放控制模块）

（4）后继业务：无

（5）功能约束：无

（6）约束描述：无

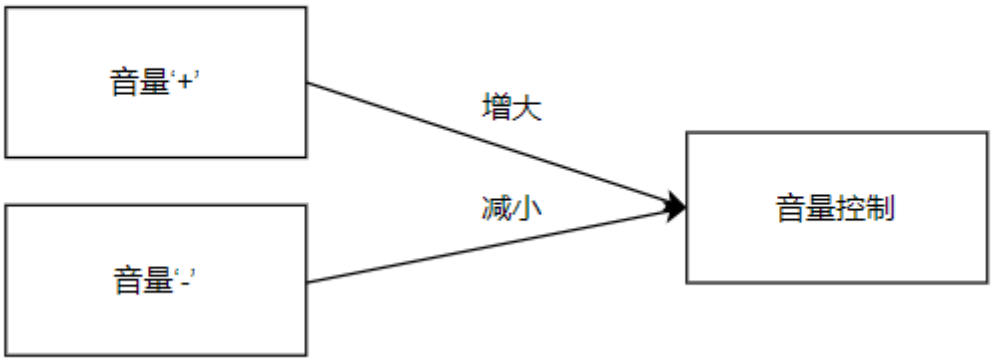
3.6.3 界面设计

3.6.4 播放列表的内部逻辑

```
def List_random():
  wmp.settings.setMode("shuffle",True)
  play()
def List_loop():
  wmp.settings.setMode("loop",True)
  play()
listimport = Button(ctr_lab,width =10,text = "导入列表")
listimport.grid(row =6,column =1,sticky ="nw",pady =5)
listexport = Button(ctr_lab,width =10,text = "导出列表")
listexport.grid(row =7,column =1,sticky = "nw",pady =5)
listdel_all = Button(ctr_lab,width =10,text = "清空列表",command = Clear_list)
```

3.7 024 音量控制模块

3.7.1 功能流程图



3.7.2 功能描述

(1) 功能类型：音量控制模块

(2) 功能描述：控制当前音量

(3) 前提业务：无

(4) 后继业务：无

(5) 功能约束：无

(6) 约束描述：无

3.7.3 界面设计

3.7.4 音量控制的内部逻辑

```
def Volume_ctr(none):
  wmp.settings.Volume = vio_scale.get()
def Volume_add(i=[0]):
  wmp.settings.Volume =wmp.settings.Volume+5
  i.append(wmp.settings.Volume)
  vio_scale.set(wmp.settings.Volume)
def Volume_minus(i=[0]):
  wmp.settings.Volume = wmp.settings.Volume -5
  i.append(wmp.settings.Volume)
  vio_scale.set(wmp.settings.Volume)
var_volume = IntVar()
vioce_lab = LabelFrame(root,text = "音量控制")
vioce_lab.grid(row =8,column =0,sticky = "wes")
vio_scale = Scale(vioce_lab,orient = HORIZONTAL,length =170,variable =
var_volume,command =Volume_ctr)
vio_scale.set(30)
vio_scale.grid(row =8,column =0)
vio_plus = Button(vioce_lab,width =8,text = "增加音量+",command =Volume_add)
vio_plus.grid(row =9,column =0,sticky = "w")
vio_minus = Button(vioce_lab,width =8,text ="减少音量-",command = Volume_minus)
vio_minus.grid(row =9,column =0,sticky ="e")
```