**面向对象的设计**

**一、词频统计需求分析**

设计词频统计是为了统计一个文档当中出现的单词数量以及出现的次数，这么做的用意是给人们提供一篇文章当中出现的重点词汇，并对文章当中的重点词汇排序，让人们更清晰的划分文章当中的词汇重点

**二、词频统计过程分析**

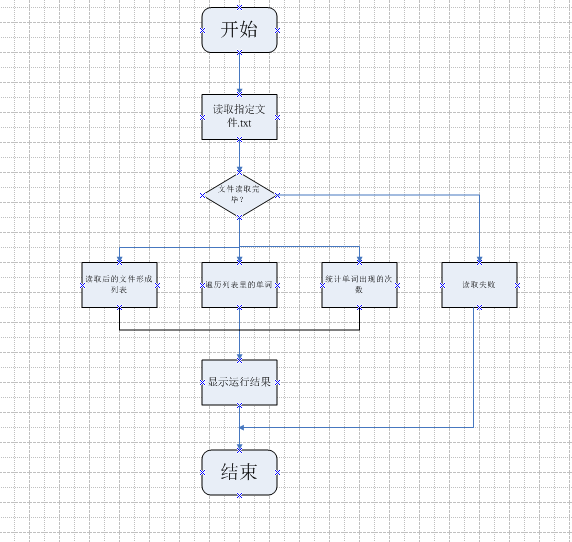
（1）程序需要读取一个文本里面的单词，可以是任意英文文本

（2）用户可以同时查询多个单词出现的次数

（3）统计文本单词及词频，并输出结果

（4）在图形界面显示文件，通过点击文件显示文件具体统计结果

**三、词频统计实现流程图**

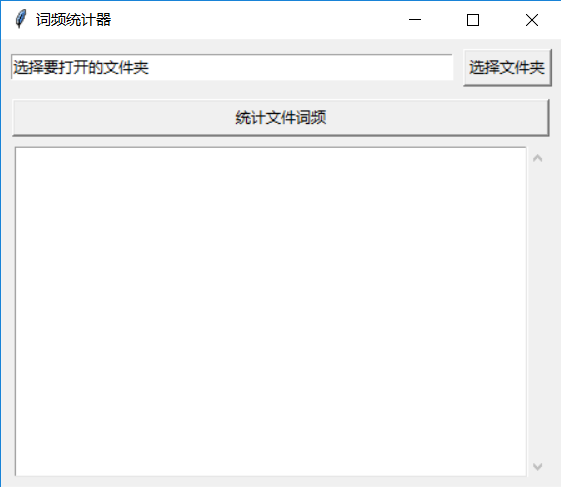


**四、词频统计界面设计及完成实现**

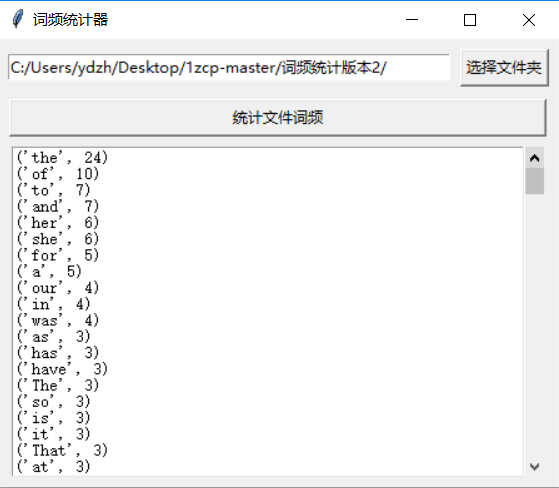
基于Python语言进行程序设计，设计一个词频统计界面窗体，窗体内容包括选择确定按钮、文件选择框及按钮、一个文本框、统计文件词频按钮；窗体完成后，成功运行后用户可以通过选择要统计词频的文件，然后点击统计文件词频后，文本框显示词频统计的结果。

1. **词频统计窗体、运行界面截图**

窗体界面：



运行结果：



1. **词频统计主要代码**

import os # 操作路径

import re # 用于描述英文单词构成

from collections import Counter # 字符统计器

from tkinter import filedialog # 文件路径对话框

from tkinter import \* # GUI

class DirectionDocWordCounter:

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

# 定义排版边距

marginx = 5

marginy = 5

# 窗口和标题

window = Tk()

window.title("词频统计器")

# 使用StringVar对象动态保管打开的文件夹位置

self.dirPath = StringVar()

self.dirPath.set("选择要打开的文件夹")

# 第一行控件打包在一个面板中

frame1 = Frame()

frame1.pack(padx=marginx, pady=(marginy, 0))

# 打包地址输入框、地址选择按钮在面板1中

Entry(frame1, width=50, textvariable=self.dirPath).pack(side=LEFT)

Button(frame1, text="选择文件夹", command=self.openDir).pack(side=LEFT, padx=(marginx, 0))

# 词频统计按钮

Button(width=60, text="统计文件词频", command=self.doCalculate).pack(padx=marginx, pady=(10,0))

# 面板2用于盛放文本域和滚动条

frame2 = Frame()

frame2.pack(padx=marginx, pady=(marginy, marginy),expand=True,fill=Y)

# 打包文本域和滚动条在面板2中

self.retText = Text(frame2, width=58, height=20, bg="white")

scrollbar = Scrollbar(frame2,orient=VERTICAL,bg="black")

self.retText.pack(side=LEFT,expand=True,fill=Y)

scrollbar.pack(side=LEFT,expand=True,fill=Y)

# 双向关联文本域和滚动条

self.retText.config(yscrollcommand=scrollbar.set)

scrollbar.config(command=self.retText.yview)

# 消息循环

window.mainloop()

def openDir(self):

# 选择要打开的文件夹路径

self.dirPath.set(filedialog.askdirectory() + "/")

def doCalculate(self):

# 统计该文件下的词频

self.countDocWordsInDir(self.dirPath.get())

# 获得单个文档词频统计的计数器对象Counter

def getFileCounter(self, filepath):

# 拿到文本

file = open(filepath, "r", encoding="gbk")

text = file.read()

# 英文单词的正则表达式

pattern = "[A-Za-z]+"

# 从text中找出所有英文单词形成列表

wordlist = re.findall(pattern, text)

# wordlist的基础上创建Counter对象

counter = Counter(wordlist)

# 返回单个文件的计数器对象

return counter

# 统计一个文件夹下所有文档的一揽子词频

def countDocWordsInDir(self, dirpath):

# 罗列一个文件下的所有文件(含文件夹)

flist = os.listdir(dirpath)

# 创建空的计数器对象，统计对象通过counter对象的加法操作逐步扩大

mcounter = Counter()

# 遍历文件夹下的所有文件

for name in flist:

# 对文档文件进行统计操作

if name.endswith(".txt"):

docpath = dirpath + name # 获得文档文件的路径

# 获得文档的计数器对象

counter = self.getFileCounter(docpath)

# 计数器对象进行加法操作，扩大统计样本

mcounter += counter

# 得到结果

resList = mcounter.most\_common()

#print(resList)

# 显示结果

for item in resList:

self.retText.insert(END, str(item) + "\n")

if \_\_name\_\_ == '\_\_main\_\_':

DirectionDocWordCounter()

pass