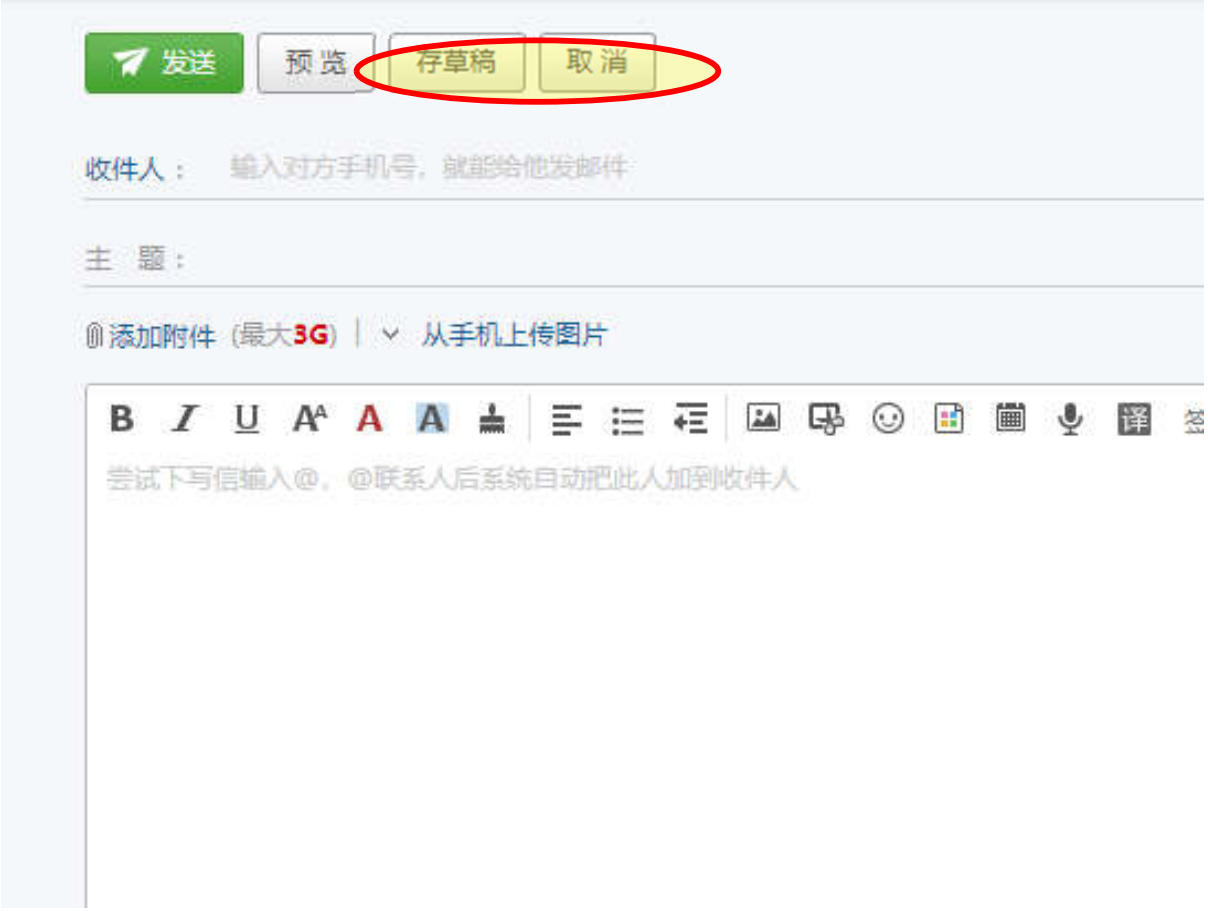
第7章数据存储及数据库程序设计

主要内容

- 邮件暂存的需求
- 文件形式的存储
- 数据库的连接
- 邮件暂存库表设计及应用
- 暂存邮件的浏览与删除

- 邮件之需求.邮件暂存、查询与删除



第7章数据存储及数据库程序设计

主要内容

- 邮件暂存的需求
- 文件形式的存储
- 数据库的连接
- 邮件暂存库表设计及应用
- 暂存邮件的浏览与删除

➢ 文件的打开与关闭

◆ 打开

文件必须存在　　文件不存在，可以创建

fileObject=open(filename,mode)

filename: 包括路径名， 如’c:\\pysrc\\msg.txt’

mode: r, w, a, rb, wb, ab, +

```
>>> f1=open('d:\\grade.txt')
>>> f1=open('d:\\grade.txt',"w+")
```

◆ 关闭

fileObject.close()

当打开文件有写入信息时，不关闭文件，在程序结束时，有可能出写文件不正确的情形。

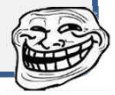
➢ 文件的打开与关闭

◆ 关闭

```
with open (filename,mode) as f:
    <利用f读写文件>
```

with中的语句执行完毕后，自动关闭文件

```
with open('d:\\grade.txt',"w+") as f:
    f.write('very good!')
```



◆ 打开异常

```
try:
    open (filename,mode)
except FileNotFoundError:
    print("……")
```

```
import os
if not os.path.exists('abc.txt'):
    print('file is not exist!')
```

➢ 向文件写入信息

【例7-1】利用文件读写函数写入字符串信息

```
try:
    with open("d:\\grade.txt",'w') as f:
        f.write('山东\n威海\n')
        L=['文化西路2号','哈尔滨工业大学(威海)']
        f.writelines(L)
        newL=[line +'\n' for line in L]
        f.writelines(newL)
except:
    print('文件创建失败')
```

➢ 从文件读取信息

【例7-2】利用文件读写函数读出字符串信息

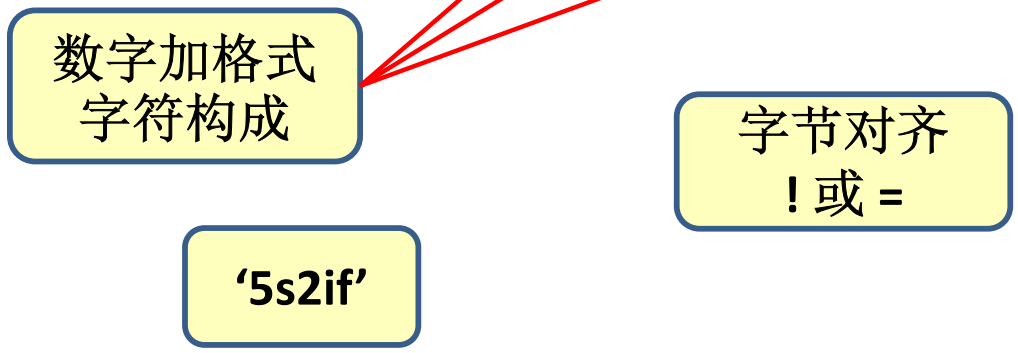
```
try:
    with open("d:\\grade.txt",'r') as f:
        s1=f.read(2)
        print(s1)
        print(f.tell())
        f.seek(6,0)
        s1=f.read(4)
        print(s1)
except FileNotFoundError:
    print('文件未发现')
```

```
try:
    with open("d:\\grade.txt",'r') as f:
        i=1
        for line in f:
            print('第',str(i),'行:',line, end="")
            i=i+1
except FileNotFoundError:
    print('文件未发现')
```

➢ 使用struct读写二进制文件

- ◆ 字节流：以字节为单位的一系列二进制数据为字节流。
- ◆ **struct 的功能：**将一系列不同类型的数据封装成一段字节流，或者相反。

(1) 数据封装函数　pack(fmt, v1, v2, ...)
(2) 数据解封函数　unpack(fmt, string)
(3) 格式大小计算函数　calcsize(fmt)



➢ 使用struct读写二进制文件

【例7-3】使用struct模块读写二进制文件

```
import struct
def m():
    with open('d:\\grade.bin','wb') as f:
        s1='张三疯'.encode('utf-8')
        s2='机械无忌'.encode('utf-8')
        byte=struct.pack("!10s12si",s1,s2,128)
        f.write(byte)
    with open('d:\\grade.bin','rb') as f:
        a,b,c=struct.unpack("!10s12si",f.read(12+18+4))
        print(a.decode("utf-8",'ignore'),b.decode("utf-8",
                                                  'ignore'),c)
m()
```

非字符串不需编码

含中文时要加

➢ 使用struct读写二进制文件

【例7-4】读取位图文件的类型及大小

```
import struct
import os
def m():
    pathfile=os.getcwd()+tes.bmp
    with open(pathfile,'rb') as f:
        a=struct.unpack(“2s",f.read(2))
        b=struct.unpack("i",f.read(4))
    print(a[0].decode(),b[0])
m()
```

➢ 使用pickle实现数据序列化

◆ 序列化：将任意python对象转化为一系列的字节存储到文件中。反序列化恰好相反

◆ 序列化对象

(1) 基本类型：布尔型、整型、浮点型、复数型等

(2) 对象：字符串、列表、元组、字典和集合等

(3) 其他：函数、类、类的实例

◆ 序列化方法

(1) **dump()** 将对象转换成字节流存入文件

dump(obj, file[,protocal])

(2) **load()**解析文件 中的字节流(反序列化)

load(file)

➢ 使用pickle实现数据序列化

【例7-5】使用pickle模块实现序列化和反序列化

```
import pickle
data={'Jack':[32,'male','market'],'Suan':[28,'female','AD.']}
L=[1,2,3]
L.insert(1,'zxd')
with open('d:\\p1.txt','wb') as f:
    pickle.dump(data,f)
    pickle.dump(L,f,1)
with open('d:\\p1.txt','rb') as f:
    data=pickle.load(f)
    print(data)
    L=pickle.load(f)
    print(L)
```

➢ 使用pickle实现数据序列化

【例7-5】使用pickle模块实现序列化和反序列化

```
import pickle
data={'Jack':[32,'male','market'],'Suan':[28,'female','AD.']}
L=[1,2,3]
L.insert(1,'zxd')
with open('d:\\p1.txt','wb') as f:
    pickle.dump(data,f)
    pickle.dump(L,f,-1)
with open('d:\\p1.txt','rb') as f:
    data=pickle.load(f)
    print(data)
    L=pickle.load(f)
    print(L)
```

===邮件暂存的需求===

- 邮件之需求.邮件暂存与删除

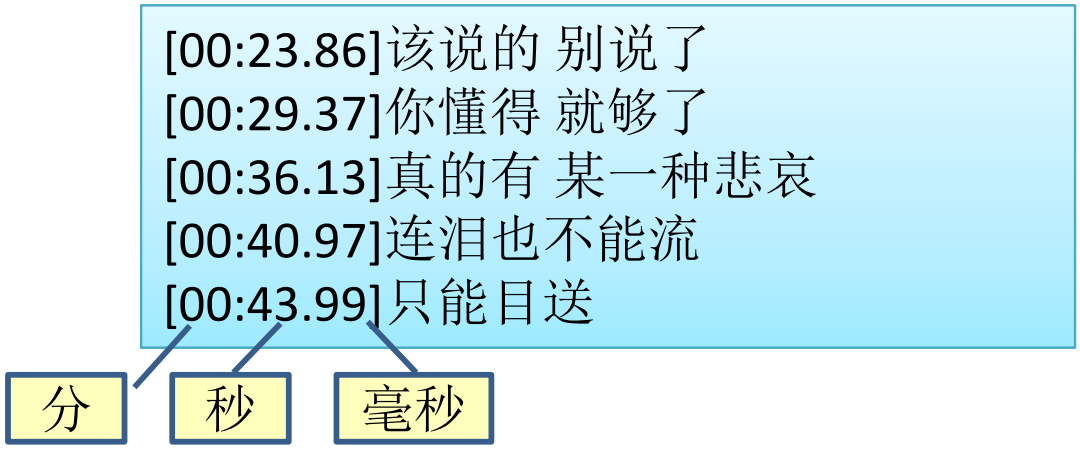

```
def savemail():
    try:
        with open("D:\\pythonemail\\mailfile.txt",'w') as f:
            f.write(var_receiver.get()+"\n")
            f.write(var_sender.get()+"\n")
            f.writelines(var_title.get()+"\n")
            f.write(t_show.get('1.0',END))
            showinfo('提示',"暂存邮件成功")
    except:
        print('文件创建失败')
        showinfo('提示',"文件创建失败")
```

- 邮件之需求.邮件暂存与删除


```
def readmail():
    try:
        with open("D:\\pythonemail\\mailfile.txt",'r') as f:
            clear()
            receiver=f.readline()
            print(receiver)
            var_receiver.set(receiver)
            sender=f.readline()
            var_sender.set(sender)
            title=f.readline()
            var_title.set(title)
            context =f.read()
            t_show.insert('1.0',context)
    except:
        print('文件读取失败')
        showinfo('提示',"文件创建失败")
```

【例7-5】读取mp3歌词文件，将时间标签转换成毫秒形式，并将每一句歌词读出来，按时间顺序以“时间（毫秒为单位）歌词”的形式显示每一句

【问题分析】mp3歌词为LRC格式，如下所示：



计算步骤：时间校验；时间转换；排序

【例7-6】读取mp3歌词文件

```
def readLRC(filename):
    with open(filename,'r',encoding='UTF-8') as f:
        res={}
        L=f.readline()
        while L!='':
            if(L[1:3].isdigit() and L[4:6].isdigit() and L[7:9].isdigit()
                and L[3]==':' and L[6]=='.'):  #校验
                t1=(int(L[1:3])*60+int(L[4:6]))*1000+int(L[7:9])*10 #转换
                res[t1]=L[10:].rstrip()
            L=f.readline()
        return res
def m():
    fname=input('输入MP3歌词文件名：')
    lrcD=readLRC(fname)
    for key in sorted(lrcD):  #排序
        print(key,lrcD[key])
m()
```

第7章数据存储及数据库程序设计

主要内容

- 邮件暂存的需求
- 文件形式的存储
- 数据库的连接
- 邮件暂存库表设计及应用
- 暂存邮件的浏览与删除

数据库的连接

- 运行环境
  PyMySQL: pip3 install PyMySQL
  安装MySQL数据库
  创建emaildb数据库
- 测试连接

```
import pymysql
# 打开数据库连接
db = pymysql.connect(host="localhost",user="root",password="密码",database="数据库" )
cursor = db.cursor() # 创建一个游标对象 cursor
cursor.execute("SELECT VERSION()") # 执行 SQL 查询
data = cursor.fetchone() # 获取单条数据.
print ("Database version : %s " % data)
db.close() # 关闭数据库连接
```

第7章数据存储及数据库程序设计

主要内容

- 邮件暂存的需求
- 文件形式的存储
- 数据库的连接
- 邮件暂存库表设计及应用
- 暂存邮件的浏览与删除

数据库的设计

◆ 数据库表（H_Email）

序号	字段	中文名	字段说明	类型与长度
1	id	用户编号	PK,主键	Integer
2	sender	发送人	NULL	VARCHAR(50)
3	receiver	接收人	NULL	VARCHAR(50)
4	context	发送内容	NULL	VARCHAR(1000)
5	c_date	创建日期	NULL	VARCHAR(19)
6	title	主题	NULL	VARCHAR(50)

◆ 创建表格

```
import pymysql
db = pymysql.connect(host="localhost",user="root",password="密码",database="数据库" )
cursor = db.cursor() #一个游标对象 cursor
# 使用预处理语句创建表
sql = """CREATE TABLE `H_Email` (
      `id` int NOT NULL AUTO_INCREMENT ,
      `sender` varchar(50) NULL , `receiver` varchar(50) NULL ,
      `context` varchar(1000) NULL ,`p_date` varchar(19) NULL ,
      `title` varchar(19) NULL , PRIMARY KEY (`id`) )"""
cursor.execute(sql)
db.close()
```

- 数据库的设计

◆ 添加数据操作

```
import pymysql
db = pymysql.connect(host="localhost",user="root",password="密码",database="数据库" )
cursor = db.cursor()
sql = """INSERT INTO h_email(sender,
        receiver, context,p_date,title)
        VALUES('hh@163.com', 'hh@hit.edu.cn',
                        '成功', '2019-03-09 21:30:01','人生')"""
try:
   cursor.execute(sql)
   db.commit()
except:
   db.rollback()
db.close()
```

```
import pymysql
db = pymysql.connect("localhost","root","123456","emaildb" )
cursor = db.cursor()
tt='2019-03-09 11:30:01'
sql="SELECT id,sender,receiver,title,context,p_date FROM h_email\
     WHERE p_date > ‘%s’  order by id" %tt  # SQL 查询语句
try:
   cursor.execute(sql)
   results = cursor.fetchall()
   for r in results:
       t_id       =r[0]
       t_sender   =r[1]
       t_receiver  = r[2]
       t_title    = r[3]
       t_context   =r[4]
       t_p_date   =r[5]
       print ("id=%s,发送人=%s,接收人=%s,主题=%s,内容=%s,\
              创建时间=%s"%\ (t_id,t_sender,t_receiver,t_title,t_context,t_p_date))
except:
   print ("Error: unable to fetch data")
db.close()
```

◆ 查询数据操作

- **fetchone():** 该方法获取下一个查询结果集。
- **fetchall():** 接收全部的返回结果行。
- **rowcount:** 返回执行execute()方法后影响的行数。

数据库的设计

◆ 更新数据操作

```
import pymysql
db = pymysql.connect("localhost","root","123456","emaildb" )
cursor = db.cursor()
tt='hh1@hit.edu.cn'
# SQL 更新语句
sql = "UPDATE h_email SET title = '必须成功' WHERE \
        receiver = '%s'" % tt
try:
   cursor.execute(sql)# 执行SQL语句
   db.commit()# 提交到数据库执行
except:
   db.rollback()# 发生错误时回滚
db.close()# 关闭数据库连接
```

数据库的设计

◆ 删除数据操作

```
import pymysql
db = pymysql.connect("localhost","root","123456","emaildb" )
cursor = db.cursor()
tt='hh1@hit.edu.cn'
# SQL 更新语句
sql = "delete from h_email  WHERE    receiver = '%s'" % tt
try:
   cursor.execute(sql)# 执行SQL语句
   db.commit()# 提交到数据库执行
except:
   db.rollback()# 发生错误时回滚
db.close()# 关闭数据库连接
```

第7章数据存储及数据库程序设计

主要内容

- 邮件暂存的需求
- 文件形式的存储
- 数据库的连接
- 邮件暂存库表设计及应用
- 暂存邮件增加、浏览与删除

- 邮件暂存与其他操作

=暂存邮件增加、浏览与删除 =

● 邮件暂存

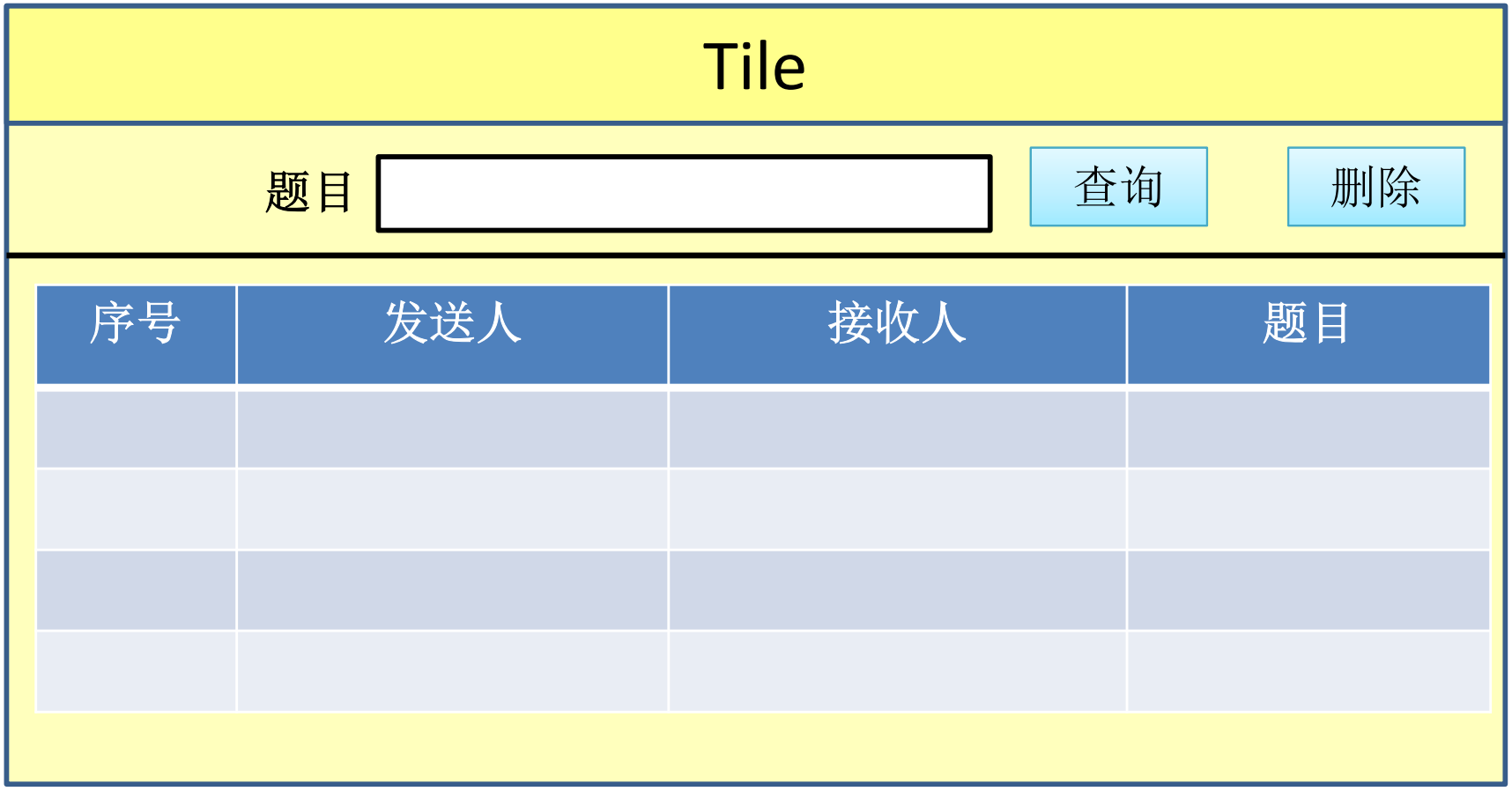
◆ 暂存操作

```
def saveEmail():
    #连接数据库
    p=DBOper("localhost","root","123456","emaildb" )
    dbsql=""insert into h_email(sender,receiver,title,context,p_date)
              values('"+s.get()+"','"
    dbsql+=t.get()+"','"+r.get()+"','"+t_show.get(0.0,tk.END)+"','"
    dbsql+=datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')+"')"
    print(dbsql)
    p.InsertDB(dbsql)
```

绑定按钮

Button(frm_S, text=“暂存”, **command=saveEmail**,width=6,
height=1, font=('Arial', 10)).pack(side=RIGHT)

- 邮件浏览界面



=暂存邮件增加、浏览与删除 =

- **grid布局**

● grid布局

◆ 创建**Toplevel**（顶级窗口）

```
top = Toplevel()
top.title('邮件暂存')
```

◆ 放置组件

```
Label(top,width=5).grid(row=0,column=0,sticky=E)
Label(top,text="题目",width=10).grid(row=0,column=1,sticky=W)
e1 = Entry(top,textvariable=v1,width=30)
e1.grid(row=0,column=2,padx=1,pady=1)
e2=Button(top,text='查询',command=lambda :search(tv,e1),width=10)
e2.grid(row=0,column=3,padx=1,pady=1)
e3=Button(top,text='删除',command=lambda :delrow(tv),width=10)
e3.grid(row=0,column=4,padx=1,pady=1)
Label(top,width=5).grid(row=0,column=5,sticky=E)
```

● treeview组件

◆ 创建treeview

```
columns = ("序号","发件人", "接收人", "题目", "发送内容","创建时间")
tv = ttk.Treeview(top, height=18, show="headings", columns=columns)
```

◆ 设置列宽及对齐方式

```
tv.column("序号", width=100, anchor='center')
tv.column("发件人", width=100, anchor='center')
tv.column("接收人", width=100, anchor='center')
tv.column("题目", width=100, anchor='center')
tv.column("发送内容", width=200, anchor='center')
tv.column("创建时间", width=200, anchor='center')
```

● treeview组件

◆ 显示表头

```
tv.heading("序号", text="序号") # 显示表头
tv.heading("发件人", text="发件人")
tv.heading("接收人", text="接收人")
tv.heading("题目", text="题目")
tv.heading("发送内容", text="发送内容")
tv.heading("创建时间", text="创建时间")
```

◆ 摆放

```
tv.grid(row=1,columnspan=6,padx=1,pady=1)
```

- 显示全部数据

```
#检索所有数据
def searchall(tv):   #tv为treeview
    #连接数据库
    p=DBOper("localhost","root","123456","emaildb" )
    #查询所有内容
    dbsql="select id,sender,receiver,title,context,p_date from h_email"
    results=p.SelDB(dbsql)
    k=0
    # 写入数据
    for row in results:
        tv.insert('', k, values=(row[0],row[1],row[2],row[3],row[4],row[5]))
        k=k+1
```

=暂存邮件增加、浏览与删除 =

- 查找数据

1.清除treeview中数据

```
def deltree(tree):
    x=tree.get_children()
    for item in x:
        tree.delete(item)
```

2. 查找数据

```
def finditem(tv,e1):
    #得到用户输入条件
    dbsql="select id,sender,receiver,title,context,p_date from h_email
            where title like '"+e1.get()+"%'"
    p=DBOper("localhost","root","123456","emaildb" )
    results=p.SelDB(dbsql)
    k=0
    for row in results:
        tv.insert('', k, values=(row[0],row[1],row[2],row[3],row[4],row[5]))
        k=k+1
```

3.显示调用

```
def search(tv,e1):
    deltree(tv)
    finditem(tv,e1)
```

4.与按钮关联

```
e2=Button(top,text='查询',command=lambda :search(tv,e1),
                width=10)
e2.grid(row=0,column=3,padx=1,pady=1)
```

- 删除数据

```
def delrow(tv):
    a=tkinter.messagebox.askokcancel('提示', '要执行此操作吗')
    if a:
        item =tv.selection()
        item_text = tv.item(item,"values")
        dbsql="delete from h_email where id='"+item_text[0]+"'"
        print(dbsql)
        #连接数据库
        p=DBOper("localhost","root","123456","emaildb" )
        p.DelDB(dbsql)
        deltree(tv)
        searchall(tv)
    else:
        return
```

```
e3=Button(top,text='删除',command=lambda :delrow(tv),width=10)
e3.grid(row=0,column=4,padx=1,pady=1)
```

=暂存邮件增加、浏览与删除 =

- 双击指令

双击后，将数据返回主界面，继续进行编辑

```
#双击函数
  def treeviewClick(event):
      print ('双击')
      for item in tv.selection():
          item_text = tv.item(item,"values")
          clear()
          s.insert(10,item_text[1])
          t.insert(10,item_text[2])
          r.insert(10,item_text[3])
          t_show.insert(1.0,item_text[4])
```

```
#绑定双击事件：<Double-1>
   tv.bind('<Double-1>', treeviewClick)
```

本章小结

- 邮件暂存的需求
- 文件形式的存储
- 数据库的连接
- 邮件暂存库表设计及数据库基本操作
- 暂存邮件增加、浏览与删除