程序设计说明

引言

编写目的

根据需求分析说明书，对系统建立起总体流程及系统总体编码规范等，为设计人员、 编程人员及测试人员工作的基础。

范围

主要针对EasyEmail项目需求分析说明书提出了基本的范围，实施目标和功能等信息，并给出有关功能的架构设计和具体方法。

背景说明

项目实施包含了

- 登录页面
- 注册页面
- 邮件页面
- 编辑页面
- 账户页面

具体架构包含了

- 网络模块
- 磁贴模块
- 分享模块
- 数据库模块
- 数据绑定模块
- 时间线程模块
- 本地文件模块
- 自适应页面模块
- 邮件处理模块
- 用户管理模块

软件环境

运行环境

客户端:
Windows 10 32bit/64bit，10240及更高版本
服务端:
CentOS 7 64bit

数据库

SQLite

开发语言

客户端:
C#
服务端:
Python

网络及硬件设备

客户端:
具有Pentium II处理器且满足以下要求的计算机：最低512MB内存，最小20GB硬盘，鼠标，键盘，有网络访问。
服务端:
具有Pentium III处理器且满足以下要求的计算机：最低256MB内存 最小20GB硬盘，有网络访问

总体概述

系统目标

构造一个轻量级聊天应用，实现以下功能
未登录用户功能:
    用户注册

用户登录

登录用户功能：

更改密码

用户登出

发送信息

查看已发送信息

接收信息

查找信息

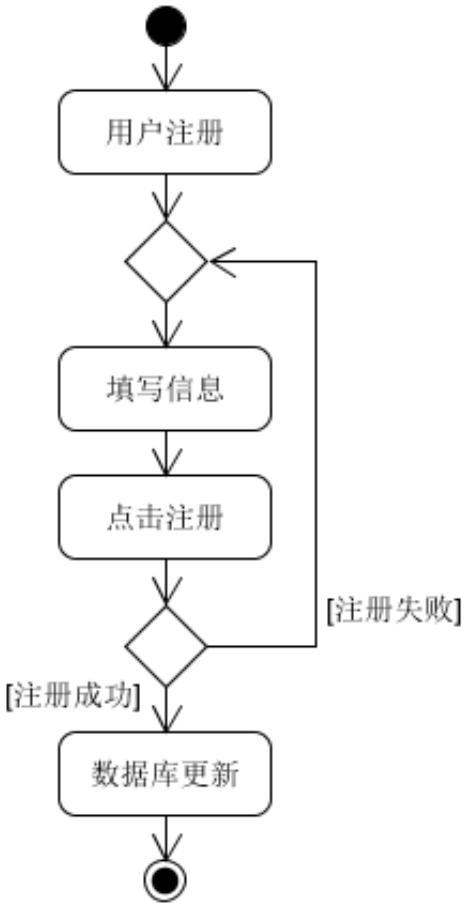
删除信息

分享信息

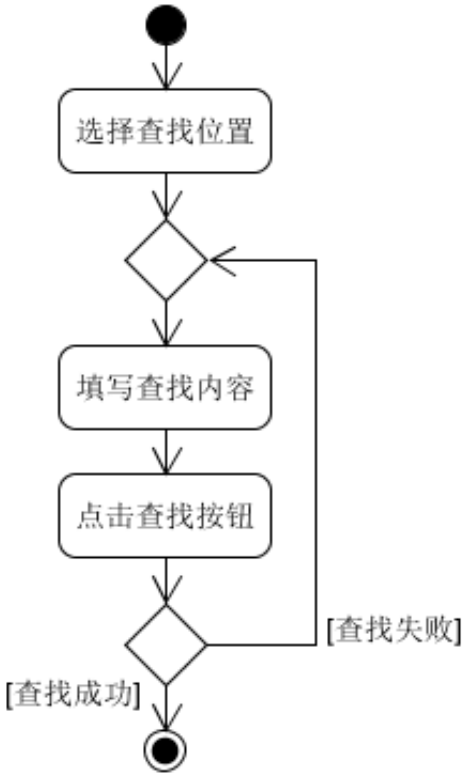
查看磁贴

程序流程图

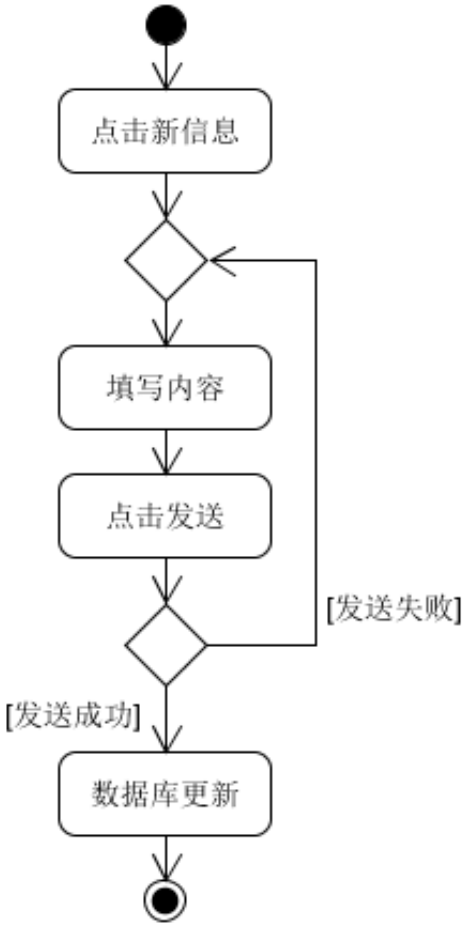
用户注册活动图



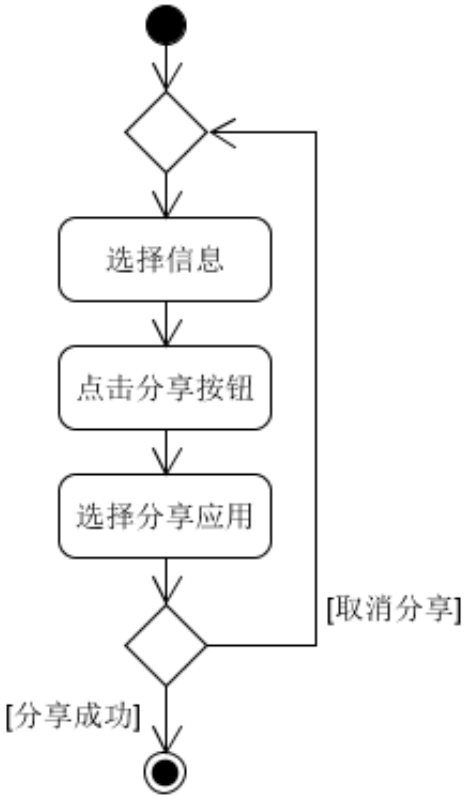
查找信息活动图



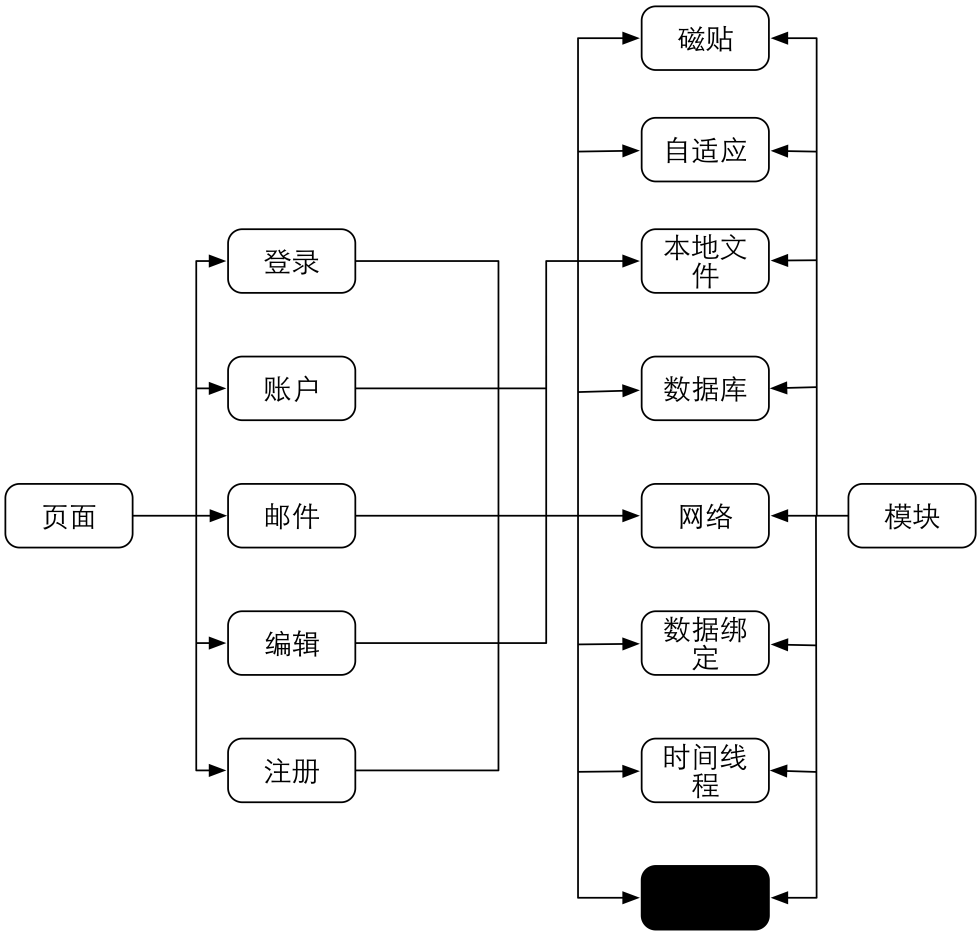
发送信息活动图



分享信息活动图



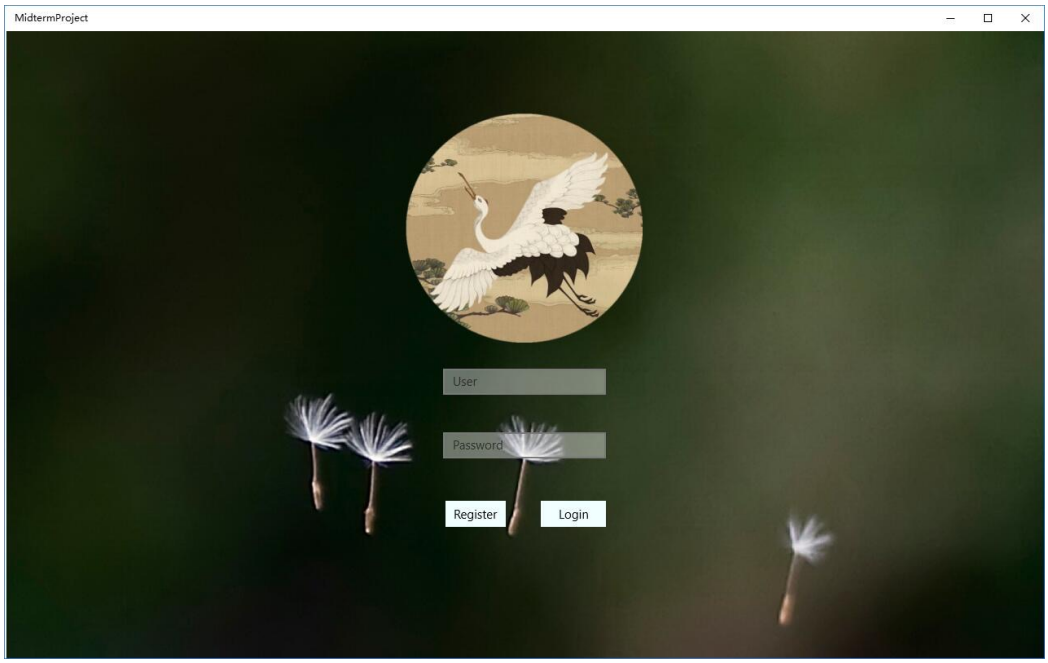
程序结构



页面功能说明

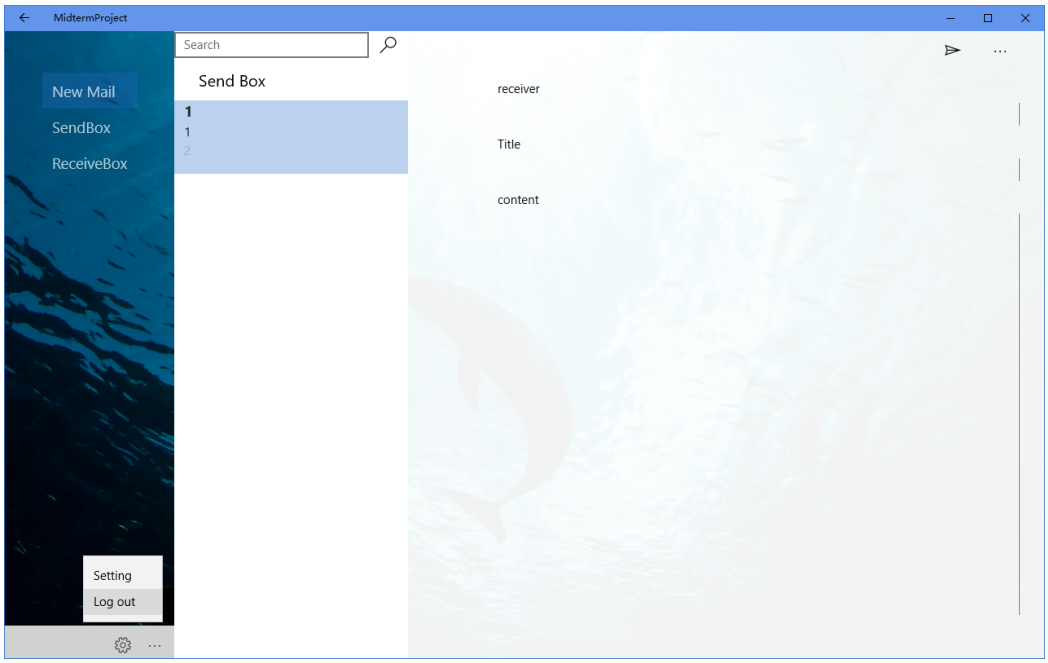
登录页面

可使用已注册的用户名和密码登录，登陆后跳转到邮件页面



注册页面
可注册未注册的用户名并设置密码

邮件页面
可收发及编辑邮件

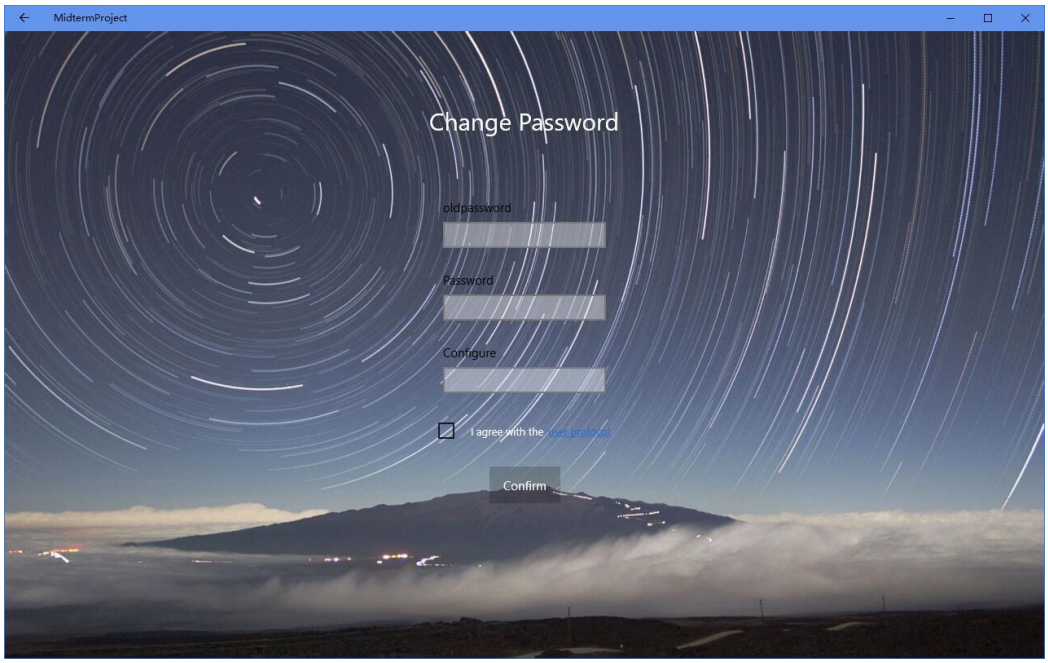


编辑页面

邮件页面宽度过小时跳转到此页面编辑邮件

账户页面

可根据已有密码更改当前用户的密码



模块分析

网络模块

使用UWP封装的HttpClient类实例同服务器建立HTTP连接，封装并交换数据

发送注册信息

```
string data =Username.Text + '\t' + Password.Password;
HttpClient httpClient = new HttpClient();
HttpResponseMessage response = await httpClient.PostAsync("
http://  sunzhongyang.com:7000/register", new StringContent
(data));
string receive = await response.Content.ReadAsStringAsync()
;
var c = new MessageDialog(receive).ShowAsync();
```

用户登录

```
string data = Username.Text + '\t' + Password.Password;
HttpClient httpClient = new HttpClient();
HttpResponseMessage response = await httpClient.PostAsync("
http://  sunzhongyang.com:7000/login", new StringContent(da
ta));
string receive = await response.Content.ReadAsStringAsync()
;
```

发送邮件

```
string data =  t.Text + "\t" + '\n' + localseetings.Values[
"user"].ToString() + '\n' +  Title.Text + '\n' + "2016" + '
\n' + Details.Text;
HttpClient httpClient = new HttpClient();
HttpResponseMessage response = await httpClient.PostAsync("
http://sunzhongyang.com:7001/send", new StringContent(data)
);
string receive = await response.Content.ReadAsStringAsync()
;
```

更改用户信息

```
string data =  localseetings.Values["user"].ToString() + '\
t' + Username.Password + '\t' + newPassword.Password;
HttpClient httpClient = new HttpClient();
HttpResponseMessage response = await httpClient.PostAsync("
http://sunzhongyang.com:7000/change", new StringContent(dat
a));
string receive = await response.Content.ReadAsStringAsync()
;
```

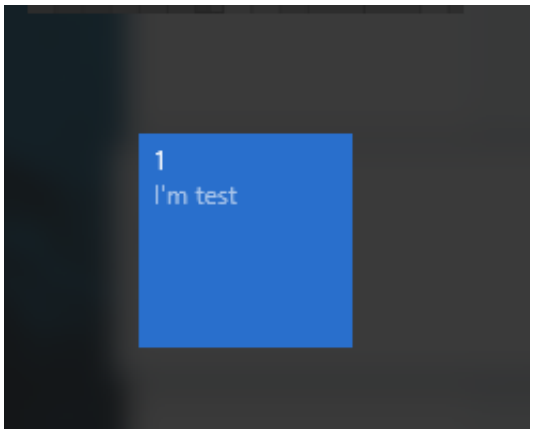
接受新邮件

```
string data = localseetings.Values["user"].ToString();
HttpClient httpClient = new HttpClient();

HttpResponseMessage response = await httpClient.PostAsync("
http://sunzhongyang.com:7001/check", new StringContent(data
));
string receive = await response.Content.ReadAsStringAsync()
;
```

磁贴模块

通过向TileUpdateManager类实例传递消息实现磁贴的更新
效果如下:




```
string from = source[0].receiver;
string subject = source[0].title;

TileContent content = new TileContent()
{
    Visual = new TileVisual()
    {
        DisplayName = "Todos",

        TileSmall = new TileBinding()
        {
            Content = new TileBindingContentAdaptive()
            {
                Children =
{
    new TileText()
    {
        Text = from
    },

    new TileText()
    {
        Text = subject,
        Style = TileTextStyle.CaptionSubtle
    },
}
```

```
            }
        },
        TileMedium = new TileBinding()
        {
            Content = new TileBindingContentAdaptive()
            {
                Children =
{
    new TileText()
    {
        Text = from
    },

    new TileText()
    {
        Text = subject,
        Style = TileTextStyle.CaptionSubtle
    },
}
            }
        },

        TileWide = new TileBinding()
        {
            Content = new TileBindingContentAdaptive()
            {
                Children =
{
    new TileText()
    {
        Text = from,
        Style = TileTextStyle.Subtitle
    },

    new TileText()
    {
        Text = subject,
        Style = TileTextStyle.CaptionSubtle
    },
}
            }
        }
    }
```

```
};

var notification = new TileNotification(content.GetXml());

TileUpdateManager.CreateTileUpdaterForApplication().Update(
notification);

//更新磁贴
if (SecondaryTile.Exists("MySecondaryTile"))
{
    var updater = TileUpdateManager.CreateTileUpdaterForSec
ondaryTile("MySecondaryTile");

    updater.Update(notification);
}
```

分享模块

通过向DataTransferManager类实例传递邮件详情实现将邮件内容分享到其他应用的功能

```
//点击分享按钮后执行的动作
private void MenuFlyoutItem_Click_1(object sender, RoutedEv
entArgs e)
{
    MenuFlyoutItem mn = (MenuFlyoutItem)e.OriginalSource;
    DataTransferManager.ShowShareUI();
}

// 实现数据请求和分享功能
void dtm_DataRequested(DataTransferManager sender, DataRequ
estedEventArgs e)
{
    string textSource = content.Text;
    string textTitle = title.Text;
    DataPackage data = e.Request.Data;
    data.Properties.Title = textTitle;
    data.SetText(textSource);
}
```

数据库模块

初始化数据库

```
string sql = @"CREATE TABLE IF NOT EXISTS
mail     (user       VARCHAR( 140 ),
          mailbox    VARCHAR( 140 ),
          sender  VARCHAR( 140 ),
          receiver     VARCHAR( 140 ),
          title     VARCHAR( 140 ),
          time     VARCHAR( 140 ),
          content     VARCHAR( 140 ));";
```

通过数据库连接实例db向数据库中写入数据

```
//显示邮件到达有关信息
var idf = new MessageDialog("You have got a mail from " + m
ail[1]).ShowAsync();
var db = App.conn;

//将收到的邮件存入数据库
var custstmt = db.Prepare("INSERT INTO mail (user, mailbox,
 sender, receiver, title, time, content) VALUES (?, ?, ?, ?
, ?, ?, ?)");

custstmt.Bind(1, localseetings.Values["user"].ToString());
custstmt.Bind(2, "receiver_box");
custstmt.Bind(3, mail[1]);
custstmt.Bind(4, mail[0]);
custstmt.Bind(5, mail[2]);
custstmt.Bind(6, mail[3]);
custstmt.Bind(7, mail[4]);
custstmt.Step();
```

通过数据库连接实例db在数据库中查询数据

```
//根据查询文本从数据库中获取需要的邮件
public string Sql_Select(string query_input)
{
    //存放查询结果的字符串
    string result = "";

    var db = App.conn;
    var statement = db.Prepare("SELECT user, mailbox, sende
r, receiver, title, time, content FROM mail WHERE user = ?
and title LIKE ? ");
    statement.Bind(1, localseetings.Values["user"].ToString
());
    statement.Bind(2, query_input);

    do
    {
        statement.Step();
        if (statement[0] == null) break;
        result += " user: " + (string)statement[0];
        result += " mailbox: " + (string)statement[1];
        result += " sender: " + (string)statement[2];
        result += " receiver: " + (string)statement[3];
        result += " title: " + (string)statement[4];
        result += " time: " + (string)statement[5];
        result += " content: " + (string)statement[6];
        result += "\n";
    } while (true);

    return result;
}
```

通过数据库连接实例db从数据库中删除数据

```
//从数据库中移除需要删除的邮件
var db = App.conn;
var statement = db.Prepare("DELETE FROM mail WHERE user = ?
 and sender = ? and title = ? and content = ? and mailbox =
 ?" );
{
    var i = new MessageDialog("Delete success").ShowAsync()
;
    statement.Bind(1, localseetings.Values["user"].ToString
());
    statement.Bind(2, this.sender.Text);
    statement.Bind(3, title.Text);
    statement.Bind(4, content.Text);
    statement.Bind(5, localseetings.Values["box"].ToString(
) == "send" ? "sender_box":"receiver_box");
    statement.Step();
}
```

数据绑定模块

通过操作实现Notify接口的数据结构实现后台数据和UI显示内容的统一

绑定数据的数据结构

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MidtermProject
{
    class Mail
    {
        public string sender { get; set; }
        public string receiver { get; set; }
        public string title { get; set; }
        public string time { get; set; }
        public string content { get; set; }

    }

}
```

绑定数据的操作接口

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.ComponentModel;

namespace MidtermProject
{
    class MailViewModel
    {
        public ObservableCollection<Mail> allMails;
        public ObservableCollection<Mail> receive_mails;
        public ObservableCollection<Mail> send_mails;
        public Mail selectmail;

        public string Addmail(int judge, string send, strin
g receiver,string title, string time, string content)
        {
```

```
            string id = Guid.NewGuid().ToString();
            switch (judge)
            {
                case 0:
                    send_mails.Insert(0,new Mail { sender =
 send, receiver = receiver, title = title, content = conten
t, time = time });
                    break;
                case 1:
                    receive_mails.Insert(0, new Mail { send
er = send, receiver = receiver, title = title, content = co
ntent, time = time });
                    break;
            }
            return id;
        }



        public MailViewModel()
        {
            allMails = new ObservableCollection<Mail>();
            receive_mails = new ObservableCollection<Mail>(
);
            send_mails = new ObservableCollection<Mail>();
            Addmail(0, "szy", "tanxiao","hello", "2015", "1
23");
            Addmail(1, "tanxiao", "szy","hi", "2016", "456"
);
        }
        public void RemoveItem()
        {
            this.allMails.Remove(this.selectmail);
            this.selectmail = null;
        }
    }


}
```

在UI中操作绑定数据实现UI内容的变化

```
//点击新邮件按钮后执行的动作
```

```
    private void New_mail_Click()
    {
        //如果此时页面宽度大于650，则显示编辑邮件栏，否则跳转到新页
面
        if (this.ActualWidth > 650)
        {
            this.show_content.Visibility = Visibility.Colla
psed;
            showmail.Visibility = Visibility.Visible;
        }
        else
        {
            Frame.Navigate(typeof(new_mail), Mailbox);
            onpage = false;
        }
    }

    //点击发件箱后执行的动作
    private void Sendbox__Click()
    {
        //设置页面状态为发件箱
        localseetings.Values["box"] = "send";

        //初始化信箱显示内容为发件箱内容
        Sql_Select_mailbox("sender_box");

        //将编辑邮件栏设置为不可见，邮件详情栏设为可见
        this.listview.Opacity = 0.8;
        this.show_content.Visibility = Visibility.Visible;
        showmail.Visibility = Visibility.Collapsed;
        show_list.Opacity = 1;
        send_or_receive.Text = "Send Box";
    }

    //点击收件箱后执行的动作
    private void Receivebox__Click()
    {
        //设置页面状态为收件箱
        localseetings.Values["box"] = "receive";

        //初始化信箱显示内容为收件箱内容
        Sql_Select_mailbox("receiver_box");
        this.listview.Opacity = 0.8;
```

```
        this.show_content.Visibility = Visibility.Visible;

        //将编辑邮件栏设置为不可见，邮件详情栏设为可见
        showmail.Visibility = Visibility.Collapsed;
        show_list.Opacity = 1;
        send_or_receive.Text = "Receive Box";
    }
```

时间线程模块

通过DispatcherTimer类实现的新增线程自动检测新邮件

```
// 创建后台线程定时器，每5秒检查一次新信息
timer = new DispatcherTimer();
timer.Tick += (s, e) => {
    if(onpage == true) check_mail();
};
timer.Interval = TimeSpan.FromMilliseconds(5000);
timer.Start();
```

本地文件模块

存储本地数据以表示页面信息 从用户名获取页面信息

```
//根据用户名检查
string data = localseetings.Values["user"].ToString();
```

从邮箱的角度获取页面信息

```
//如果当前视图正在显示收件箱中的邮件，则更新收件箱视图
if (localseetings.Values["box"].ToString() == "receive")
{
    Sql_Select_mailbox("receiver_box");
}
```

自适应页面模块

根据窗口大小获得页面的现实逻辑

```
private void reply_Click(object sender, RoutedEventArgs e)
{
    //获取回复邮件的收件人
    string receive_user = this.sender.Text;

    //根据当前窗口大小决定是否跳转到新页面完成邮件的写作
    if (this.ActualWidth > 650)
    {
        this.show_content.Visibility = Visibility.Collapsed
;
        showmail.Visibility = Visibility.Visible;
        t.Text = receive_user;
    }
    else
    {
        Frame.Navigate(typeof(new_mail), receive_user);
        onpage = false;
    }
}
```

邮件处理模块

收到邮件并存储在数据结构中，需要时可按规则发送给用户

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import textwrap

import tornado.httpserver
import tornado.ioloop
import tornado.options
import tornado.web

#设置接收端口为7001
from tornado.options import define, options
define("port", default=7001, help="run on the given port",
type=int)

#设置邮件收件人为键，邮件内容为值的字典
receiver_mailcontent = {}
```

```
#处理发件请求的类
class sendHandler(tornado.web.RequestHandler):
    def post(self):

        #获取报文
        result = ""
        text = self.request.body
        text = str(text)

        #从报文中取出收件名和邮件内容
        receiver, mailcontent = text.split('\t')

        #设置词典中对应收件人的内容
        receiver_mailcontent[receiver] = mailcontent

        #写入成功报文
        result = "success"
        self.write(result)

class checkHandler(tornado.web.RequestHandler):
    def post(self):

        #获取报文
        result = ""
        text = self.request.body
        text = str(text)

        #确定字典中是否有制定的用户名所对应的邮件，如果没有则返回No
        if receiver_mailcontent.has_key(text):
            if receiver_mailcontent[text] != "":
                result = receiver_mailcontent[text]

                #获取返回邮件后将字典中收件人对应的邮件置空
                receiver_mailcontent[text] = ""
            result = "No"
        result = "No"

        self.write(result)

if __name__ == "__main__":
    tornado.options.parse_command_line()
    app = tornado.web.Application(
        handlers=[
```

```
            (r"/send", sendHandler),
            (r"/check",checkHandler)
        ]
    )
    http_server = tornado.httpserver.HTTPServer(app)
    http_server.listen(options.port)
    tornado.ioloop.IOLoop.instance().start()
```

用户管理模块

存储用户基本信息并提供验证服务

```
    #!/usr/bin/python
# -*- coding: UTF-8 -*-

import textwrap

import tornado.httpserver
import tornado.ioloop
import tornado.options
import tornado.web

#设置接收端口为7000
from tornado.options import define, options
define("port", default=7000, help="run on the given port",
type=int)

#设置用户名为键，用户密码为值的字典
user_password = {}

#处理登陆请求的类
class loginHandler(tornado.web.RequestHandler):
    def post(self):

        #获取报文
        result = ""
        text = self.request.body
        text = str(text)

        #从报文中取出用户名和密码
        username, password = text.split('\t')
```

```
        #根据正确与否决定返回报文
        if user_password.has_key(username):
            if user_password[username] == password:
                result = "Login success"
            else:
                result = "Error password"
        else:
            result = "Error user"
        self.write(result)

#处理注册请求的类
class registerHandler(tornado.web.RequestHandler):
    def post(self):
    result = ""
    text = self.request.body
    text = str(text)
    username, password = text.split('\t')
    if user_password.has_key(username):
        result = "User exist"
    else:
        user_password[username] = password
        result = "register success"
    self.write(result)

#处理变更密码请求的类
class changeHandler(tornado.web.RequestHandler):
    def post(self):

        #获取报文
        result = ""
        text = self.request.body
        text = str(text)

        #从报文中获取用户名和新旧密码
        username, oldpassword, newpassword = text.split('\t
')

        #根据密码的正确与否决定是否更改原密码，并返回指定内容
        if user_password.has_key(username):
        if user_password[username] == oldpassword:
            user_password[username] = newpassword
            result = "change success"
        else:
```

```
            result = "oldpassword wrong"
    else:
        result = "No account"
    self.write(result)

if __name__ == "__main__":
    tornado.options.parse_command_line()
    app = tornado.web.Application(
        handlers=[
            (r"/login", loginHandler),
            (r"/register",registerHandler),
        (r"/change", changeHandler)
        ]
    )
    http_server = tornado.httpserver.HTTPServer(app)
    http_server.listen(options.port)
    tornado.ioloop.IOLoop.instance().start()
```

数据库设计

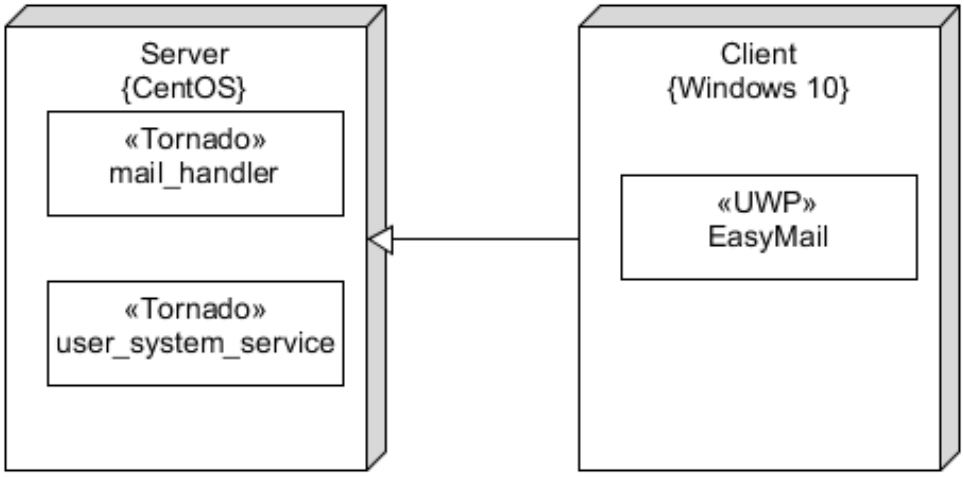
数据库采用sqlite方式，sqlite是一个本地的轻型的关系型数据库，支持一般的sql操作
在本程序中数据库主要用于在本地存储用户收发的邮件，并提供搜索邮件的接口，数据的具体存储格式如下：

```
user       VARCHAR( 140 )
mailbox    VARCHAR( 140 )
sender  VARCHAR( 140 )
receiver     VARCHAR( 140 )
title     VARCHAR( 140 )
time     VARCHAR( 140 )
content     VARCHAR( 140 )
```

所有数据均为VARCAR格式，user为用户名，mailbox对应收件箱或者发件箱，sender是发件人，receiver是收件人，title是邮件标题，time是发件时间，content为邮件具体内容

部署架构

如图所示


Server
{CentOS}
«Tornado»
mail_handler
«Tornado»
user_system_service
Client
{Windows 10}
«UWP»
EasyMail