

SMTP 服务器实验报告

1711437 任蕾

目录

1 实验目的	2
2 实验环境	2
3 实验内容	2
3.1 预备工作	2
3.2 初始化对话框界面	3
3.3 建立连接	4
3.3.1 绑定 IP 和端口号	4
3.3.2 接收连接	4
3.4 邮件传递	5
3.4.1 命令交互	5
3.4.2 邮件处理	8
3.5 连接关闭	16
3.6 额外功能实现	17
4 实验结果	21
4.1 初始界面	21
4.2 发送邮件	21
4.3 通信过程日志及接收到的邮件内容	22
4.4 带格式邮件内容显示	23
4.5 附件显示及存储	24
5 实验总结	25
5.1 SMTP 邮件的传递过程	25
5.2 电子邮件报文的组成	25
5.3 多用 Internet 邮件扩展协议 MIME	25
5.4 基数 64 编码 (base64) 编码算法	26

1 实验目的

观察电子邮件应用程序与 SMTP 邮件服务器的命令交互过程。

2 实验环境

mac 上使用 qt 编写实现，mac 自带的 mail 邮箱作为邮件客户端。

3 实验内容

3.1 预备工作

SMTP 服务器在 TCP 的 25 端口守候，这里我修改本机邮件客户端 smtp 服务器设置为端口 2019。





3.2 初始化对话框界面

清空图片框、初始化对话框界面。

```

Dialog::Dialog(QWidget *parent)
    : QDialog(parent)
    , ui(new Ui::Dialog)
{
    ui->setupUi(this);

    // 初始化
    tcpServer = new QTcpServer();
    tcpSocket = nullptr;
    data = "";
    ui->img->clear();

    ui->logShow->append("*** SMTP服务器准备好");
    ui->logShow->append("$$$$$$$$$$$$$$$$$$$$");
}

```

3.3 建立连接

3.3.1 绑定 IP 和端口号

使用 listen() 监听同时绑定 IP 和端口号，如果失败，弹出错误信息对话框。

```

// 调用listen函数监听同时绑定IP和端口号
if(tcpServer->listen(QHostAddress::LocalHost,2019))
{
    // 将服务器的连接信号连接到接收连接的槽
    this->connect(tcpServer,SIGNAL(newConnection()),this,SLOT(ready_acpt()));
}
else
{
    QMessageBox::critical(this, tr("网络"),
                         tr("错误: %1.").arg(tcpServer->errorString()));
}

```

3.3.2 接收连接

查询到有新连接，获取到客户端套接字，则发送 220 信息回复客户端本服务器状态。

```

// 建立连接
void Dialog::ready_acpt()
{
    ui->logShow->append("**** 收到连接请求");
    ui->logShow->append("**** 建立连接");
    if(tcpServer->hasPendingConnections()) // 查询是否有新连接
    {
        // 获取客户端套接字
        tcpSocket = tcpServer->nextPendingConnection();
        if(tcpSocket != nullptr) // 客户端存在
        {
            // 连接成功，回复本服务器状态
            QString msg = "220 Simple Mail Server Ready for Mail Server";
            ui->logShow->append("S:" + msg);
            // 使用write函数向服务器发送数据
            QString sendmsg = msg + "\r\n";
            tcpSocket->write(sendmsg.toLocal8Bit());
            // 接收消息
            this->connect(tcpSocket,SIGNAL(readyRead()),this,SLOT(recv_msg()));
            // 连接结束
            this->connect(tcpSocket,SIGNAL(disconnected()),this,SLOT(cnt_end()));
        }
    }
}

```

3.4 邮件传递

3.4.1 命令交互

常用的 SMTP 命令

命令	描述
HELO <主机域名>	开始会话
MAIL FROM: <发送者电子邮件地址>	开始一个邮递处理，指出邮件发送者
RCPT TO: <接收者电子邮件地址>	指出邮件接收者
DATA	接收程序将 DATA 命令后面的数据作为 邮件内容处理，直到<CR><LF>.<CR> <LF>出现
RSET	中止当前的邮件处理
NOOP	无操作
QUIT	结束会话

常用的 SMTP 响应

响应以 3 位数字开始，后面跟有该响应的具体描述。

命令	描述
220	域服务准备好
221	系统状态或系统帮助应答
250	请求的命令成功完成
354	可以发送邮件内容
500	语法错误，命令不能识别
502	命令未实现
550	邮箱不可用

接收客户端发来的消息。

- 1) 若接收到 SMTP 命令 HELO 或者 EHLO，发送 250 消息给客户端，开始会话，并将消息交互显示在服务器日志界面；
- 2) 若接收到 SMTP 命令 MAIL FROM、RCPT TO，表示开始邮件处理，指出邮件接受者及接受者，发送 250 消息给客户端，并将消息交互显示在服务器日志界面；
- 3) 若接收到 SMTP 命令 DATA，表示之后的数据是邮件内容，直到 <CR><LF>.<CR><LF> 出现表示结束，发送 334 消息给客户端，开始接收邮件内容，完成之后回复 250 消息给客户端，将邮件数据交给邮件处理函数，并将消息交互显示在服务器日志界面；
- 4) 若接收到 SMTP 命令 RSET，则中止当前邮件处理，并将消息交互显示在服务器日志界面；
- 5) 若接收到 SMTP 命令 QUIT，表示结束回话，回复 221 消息，并将消息交互显示在服务器日志界面。

```
void Dialog::recv_msg()
{
    // 获得消息
    QByteArray buffer = tcpSocket->readAll();
    QString recvmsg = QString::fromLocal8Bit(buffer);
    if(recvmsg.left(4) == "HELO" || recvmsg.left(4) == "EHLO")
    {
        ui->logShow->append("C:" + recvmsg.trimmed()); // 去掉首尾空白符
        QString msg = "250 OK 127.0.0.1";
        ui->logShow->append("S:" + msg);
        // 使用write函数向服务器发送数据
        QString sendmsg = msg + "\r\n";
        tcpSocket->write(sendmsg.toLocal8Bit());
    }
    else if(recvmsg.left(10) == "MAIL FROM:")
    {
        ui->logShow->append("C:" + recvmsg.trimmed());
        QString msg = "250 Sender OK";
        ui->logShow->append("S:" + msg);
        // 使用write函数向服务器发送数据
        QString sendmsg = msg + "\r\n";
        tcpSocket->write(sendmsg.toLocal8Bit());
    }
    else if(recvmsg.left(8) == "RCPT TO:")
    {
        ui->logShow->append("C:" + recvmsg.trimmed());
        QString msg = "250 Receiver OK";
        ui->logShow->append("S:" + msg);
        // 使用write函数向服务器发送数据
        QString sendmsg = msg + "\r\n";
        tcpSocket->write(sendmsg.toLocal8Bit());
    }
    else if(recvmsg.left(4) == "DATA")
    {
        ui->logShow->append("C:" + recvmsg.trimmed());
        QString msg = "354 Go ahead. End with <CRLF>.<CRLF>";
        ui->logShow->append("S:" + msg);
        // 使用write函数向服务器发送数据
        QString sendmsg = msg + "\r\n";
        tcpSocket->write(sendmsg.toLocal8Bit());
    }
}
```

```

else if(recvmsg.left(4) == "QUIT")
{
    ui->logShow->append("C:" + recvmsg.trimmed());
    QString msg = "221 Quit, Goodbye !";
    ui->logShow->append("S:" + msg);
    // 使用write函数向服务器发送数据
    QString sendmsg = msg + "\r\n";
    tcpSocket->write(sendmsg.toLocal8Bit());
}
else if(recvmsg.left(4) == "RSET"){
    ui->logShow->append("C:" + recvmsg.trimmed());
}
else // 邮件DATA内容
{
    data = data + recvmsg;
    if(data.right(5) == "\r\n.\r\n")
    {
        showMail(data);
        QString msg = "250 Message accepted for delivery";
        ui->logShow->append("S:" + msg);
        // 使用write函数向服务器发送数据
        QString sendmsg = msg + "\r\n";
        tcpSocket->write(sendmsg.toLocal8Bit());
    }
}
}

```

3.4.2 邮件处理

邮件头和邮件体之间用空行分隔，因此先将邮件数据按行进行分割，判断空行以分离开邮件头和邮件体及每部分内容。

```

ui->mailShow->append("$$$$$$$$$$$$$$$$$$$$");
// ui->mailShow->append(mail_data);

QStringList data_list = mail_data.split("\r\n");

```

处理邮件头，将每一行数据输出到邮件显示窗口，如果行内容判断是主题，则检查主题编码方式，如果是=? 和?= 包括的内容，则表明有不同编码，中间的?是间隔符，第一段表明原来页面的编码方式 utf-8 或者 gbk；第二段如果是B，则表示邮件的编码方式为 base64；第三段是标题内容，按照相应方式进行解码即可。

```
// 处理邮件头
for(i = 0;fl == 0;i++)
{
    QString tmp = data_list.at(i); // 一行数据
    if(tmp.isEmpty()) // 邮件头和邮件体之间用空行分隔
    {
        fl = 1; // 标志空行，分离每一大段
    }
    else if(tmp == "Content-Transfer-Encoding: base64")
    {
        ec_type = 1;
        mail_head.append(tmp + '\n');
    }
    else if(tmp.left(23) == "Content-Type: multipart")
    {
        ec_type = 2;
        mail_head.append(tmp + '\n');
    }
    // 分了3段，=?和?=是开始和结束标记，中间的?是间隔符
    // 第一段：utf-8或者gbk，表示原来页面的编码方式
    // 第二段：B是表示邮件的编码方式为base64
    // 第三段：标题内容
    else if(tmp.left(11) == "Subject: =?")
    {
        QStringList tp = tmp.split('?', true);
        if(tp.at(2) == "B")
        {
            mail_head.append("Subject: " + base64_decode(tp.at(3)) + '\n');
        }
        else
        {
            mail_head.append(tmp + '\n');
        }
    }
    else
    {
        mail_head.append(tmp + '\n');
    }
}
```

处理邮件体，如果邮件头中出现 Content-Transfer-Encoding: base64，则表明邮件是简单 base64 位编码的，如果出现 Content-Type: multipart，则表示邮件中有多种编码。

简单邮件

对于每一行数据，比对键值部分数据，按照数据类型及编码方式进行不同处理。

```
// 处理邮件体
if(ec_type == 0) //简单7bit编码，或quoted-printable编码（暂未处理）
{
    for(;i < data_list.size() - 2;i++)
    {
        QString tmp = data_list.at(i);
        ui->mailShow->append(tmp);
    }
}
else if(ec_type == 1) //简单base64编码邮件
{
    for(;i < data_list.size() - 2;i++)
    {
        QString tmp = data_list.at(i); // 一行数据
        if(tmp.isEmpty())
        {
            ui->mailShow->append(tmp);
        }
        else
        {
            mail_body.append(tmp + "\r\n");
        }
    }
    QString re_body = base64_decode(mail_body);
    ui->mailShow->append(re_body);
}
```

复杂邮件

1) 对于每一行数据，通过对每行开头部分说明的读取判断每部分的数据格式和编码方式以及是否为附件，不同编码方式用不同解码函数处理，不同文件格式也分开处理。

```
ui->mailShow->append(data_list.at(i));
int flag = 0,type_fl = 0,has_e = 0; //标志base64/文件格式/是否有附件
QString f_name,f_type; // 记录储存文件的名称和格式
for(i += 1;i < data_list.size() - 2;i++)
{
    QString tmp = data_list.at(i);
    if(tmp == "Content-Transfer-Encoding: base64")
    {
        flag = 1;
        ui->mailShow->append(tmp);
    }
    else if(tmp == "Content-Transfer-Encoding: quoted-printable")
    {
        flag = 2;
        ui->mailShow->append(tmp);
    }
    else if(tmp.left(19) == "Content-Type: image")
    {
        type_fl = 1;
        // QStringList ft = tmp.split('/');
        // f_type = ft.at(1).left(ft.length()-1);
        ui->mailShow->append(tmp);
    }
    else if(tmp.left(23) == "Content-Type: text/html")
    {
        type_fl = 2;
        ui->mailShow->append(tmp);
    }
}
```

如果是附件，还需要判断附件名是否有编码，并对编码的附件名进行解码。

```
else if(tmp.left(6) == "\tname=") // 有附件
{
    has_e = 1;
    QStringList t = tmp.split('\'');
    if(t.at(1).startsWith("=?"))
    {
        QStringList tp = tmp.split('?');
        if(tp.at(2) == "B")
        {
            QByteArray fn = base64_decode(tp.at(3));
            ui->mailShow->append("\tname=\"" + fn + "\"");
            f_name = fn.toStdString().c_str();
        }
        else
        {
            ui->mailShow->append(tmp);
        }
    }
    else
    {
        f_name = t.at(1);
        ui->mailShow->append(tmp);
    }
}
else if(tmp.left(9) == "\tfilename")
{
    QStringList tp;
    if(tmp.contains("utf-8"))
    {
        tp = tmp.split("\'\'");
        ui->mailShow->append("\tfilename=" + tp.at(1));
    }
    else
    {
        ui->mailShow->append(tmp);
    }
}
```

2) 对于图片格式的附件，使用 `base64_decode()` 或者 `quoted_decode()` 函数解码之后显示在相应区域，即可。

3) 对于有格式的 html 邮件内容，解码后可将纯文本内容输出到相应区域，再将原本的 html 格式显示的内容输出。

```
else if(tmp.isEmpty()) // 每部分邮件体之间用空行分隔
{
    ui->mailShow->append(tmp);
    i++;
    tmp = data_list.at(i);
    QString part_body;
    while(tmp.left(12) != "--Apple-Mail")
    {
        part_body.append(tmp + "\r\n");
        i++;
        tmp = data_list.at(i);
    }
    if(flag == 1) // 是base64编码的部分
    {
        // ui->mailShow->append("原: " + part_body);
        QByteArray re_body = base64_decode(part_body);

        if(type_fl == 1) // 图片
        {
            ui->mailShow->append("图片见左图。");
            ui->img->clear();

            QBuffer buf(&re_body);
            buf.open(QIODevice::WriteOnly);
            QImage image; // 用QImage进行加载, 然后转乘QPixmap用户绘制。
            if(!image.loadFromData(re_body))
            {
                ui->mailShow->append("image loaded wrong. ");
            }
            ui->img->setPixmap(QPixmap::fromImage(image));
            image.save("/Users/renlei/Qt-workspace/img/" + f_name);
            type_fl = 0;
        }
        else if(type_fl == 2) // html
        {
            ui->mailShow->append(re_body);

            type_fl = 0;
        }
        else
        {
            ui->mailShow->append(re_body);
        }
        flag = 0;
}
```

不是这两种编码方式的内容直接输出。

```
else // 不是base64和quoted
{
    ui->mailShow->append(part_body);
    type_fl = 0;
}
```

base64 解码实现

解码表如下。

```
// 解码表, 128个ascii编码对应
static const unsigned char base64_decode_table[128] =
{
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xE0, 0xF0, 0xFF, 0xF1, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xE0, 0xFF, 0xFF,
    0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0xFF, 0xFF, 0x00, 0xFF, 0xFF,
    0xFF, 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E,
    0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18, 0x19, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xFF, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28,
    0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32, 0x33, 0xFF, 0xFF, 0xFF, 0xFF
};
```

解码过程就是编码的逆过程。

```
// base64解码
QByteArray Dialog::base64_decode(QString dt)
{
    // QByteArray test = QByteArray::fromBase64(data.toUtf8());
    // QString res = test.toStdString().c_str();
    // return res;

    if(dt.isEmpty())
    {
        return dt.toLocal8Bit();
    }
    // ui->mailShow->append(data);

    int len = dt.length(); // 字符串长度

    QByteArray res;
    QDataStream out(&res, QIODevice::WriteOnly); // 写入数据流

    int ind = 0;
    int p = 0;
    for(int i = 0;i < len; i++)
    {
        int v = dt[i].unicode();
        if(v >= 0x80) // ascii最大0x7F
        {
            ind = -1;
            break;
        }
        int value = base64_decode_table[v]; // 找到对应编码
        if(value == 0xFF)
        {
            ind = -1;
            break;
        }

        // 跳过换行\n回车\r连字符-制表符\t空格space
        if(value == 0xF0||value == 0xF1||value == 0xF2||value == 0xE0||value == 0xF3)
            continue;

        if(ind == 0)
        {
            if(v == '=')
            {
                ind = -1;
                break;
            }

            p = value;
            ind++;
        }
    }
}
```

```
else if(ind == 1)
{
    if(v == '=')
    {
        ind = -1;
        break;
    }

    out<<static_cast<unsigned char>((p<<2)|(value>>4));
    p = value;
    ind++;
}
else if(ind == 2)
{
    if(v == '=')
    {
        ind = 0;
        break;
    }

    out<<static_cast<unsigned char>((p<<4)|(value>>2));
    p = value;
    ind++;
}
else
{
    ind = 0;
    if(v == '=')
        break;

    out<<static_cast<unsigned char>((p<<6)|value);
}

if(ind == 0) // 有效字符个数必须是4的倍数
    return res;
else if(ind == -1)
    return "base64_decode wrong1";
else
    return "base64_decode wrong2";
}
```

3.5 连接关闭

完成交互后关闭连接。

```
// 关闭连接
void Dialog::cnt_end()
{
    if(tcpSocket != nullptr)
    {
        ui->logShow->append("关闭连接");
        tcpSocket->close();
        tcpSocket->deleteLater();
        //tcpServer->close();
    }
}
```

3.6 额外功能实现

1) base64 编码实现

```
// base64编码，3个字节转换成4个可打印字符
QByteArray Dialog::base64_encode(QString dt)
{
    if(dt.isEmpty())
    {
        return dt.toLocal8Bit();
    }

    QString res;
    QByteArray tmp = dt.toLocal8Bit();
    int ind = 0;
    for(int i = tmp.length(); i > 0; i -= 3)
    {
        if(i >= 3)
        {
            int b0 = tmp.at(ind) & 0xFF;
            int b1 = tmp.at(ind+1) & 0xFF;
            int b2 = tmp.at(ind+2) & 0xFF;
            res.append(base64_encode_table[b0>>2]);
            res.append(base64_encode_table[((b0<<4)|(b1>>4)) & 0x3F]);
            res.append(base64_encode_table[((b1<<2)|(b2>>6)) & 0x3F]);
            res.append(base64_encode_table[b2 & 0x3F]);
            ind += 3;
        }
    }
}
```

```
    else
    {
        int b0 = tmp.at(ind) & 0xFF;
        int b1;
        if(i == 2)
        {
            b1 = tmp.at(ind+1) & 0xFF;
        }
        else
        {
            b1 = 0;
        }
        res.append(base64_encode_table[b0>>2]);
        res.append(base64_encode_table[((b0<<4)|(b1>>4)) & 0x3F]);
        if(i == 1)
        {
            res.append('=');
        }
        else
        {
            res.append(base64_encode_table[(b1<<2) & 0x3F]);
        }
        res.append('=');
    }
}
return res.toLocal8Bit();
}
```

2) quoted 解码实现

```
QByteArray Dialog::quoted_decode(QString dt)
{
    QByteArray dt_tmp = dt.toLocal8Bit();
    int i = 0;
    char tmp = 0;
    QByteArray res;

    while(i < dt.length())
    {
        if(dt.at(i) == '=' && dt.at(i+1) == '\r' && dt.at(i+2) == '\n')
        {
            i += 3;
        }
        else if(dt.at(i) == '=')
        {
            char c1 = dt_tmp.at(i+1);
            char c2 = dt_tmp.at(i+2);
            tmp = ((c1>'9')?(c1-'A'+10):(c1-'0'))*16 + ((c2>'9')?(c2-'A'+10):(c2-'0'));
            res.append(tmp);
            i += 3;
        }
        else
        {
            res.append(dt.at(i));
            i++;
        }
    }

    return res;
}
```

3) 中文支持

QT 默认的编码 (unicode) 是不能显示中文的, 可以用 QString 的 toLocal8bit() 与 fromLocal8bit() 以及字符先转 unicode 编码再进行 base64 解码等其他操作来解决。

```
QByteArray dt_tmp = dt.toLocal8Bit();
int i = 0;
char tmp = 0;
QByteArray res;
```

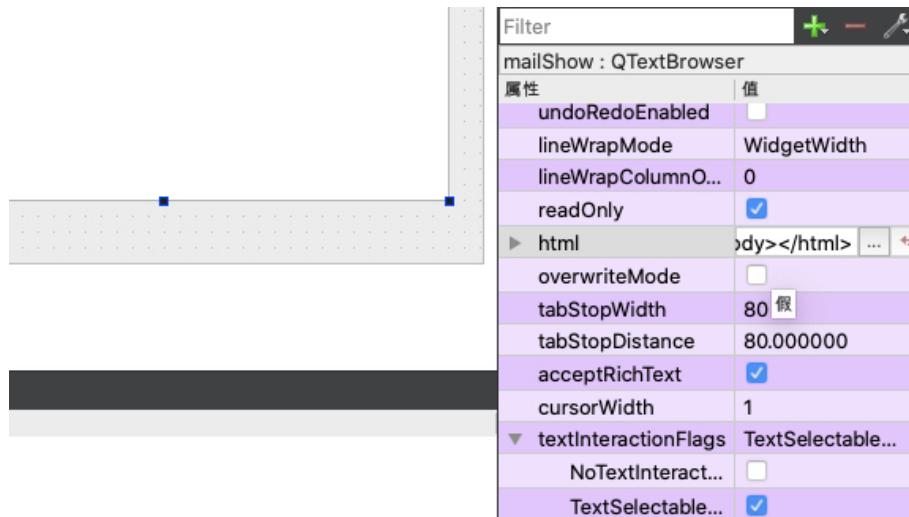
```

int ind = 0;
int p = 0;
for(int i = 0; i < len; i++)
{
    int v = dt[i].unicode();
    if(v >= 0x80) // ascii最大0x7F
    {
        ind = -1;
        break;
    }
}

```

4) 带格式邮件内容带格式输出

将获得的数据相应解码，使用 Text Browser 文本控件输出。



5) 附件保存

将接收到的数据解码下载，按解析所得的附件名进行存储，文件存储同图片类似。

```

QBuffer buf(&re_body);
buf.open(QIODevice::WriteOnly);

QImage image; // 用QImage进行加载，然后转乘QPixmap用户绘制。QPixmap绘制效果是最好的。
if(!image.loadFromData(re_body))
{
    ui->mailShow->append("image loaded wrong. ");
}
ui->img->setPixmap(QPixmap::fromImage(image));
image.save("/Users/renlei/Qt-workspace/img/" + f_name);

```

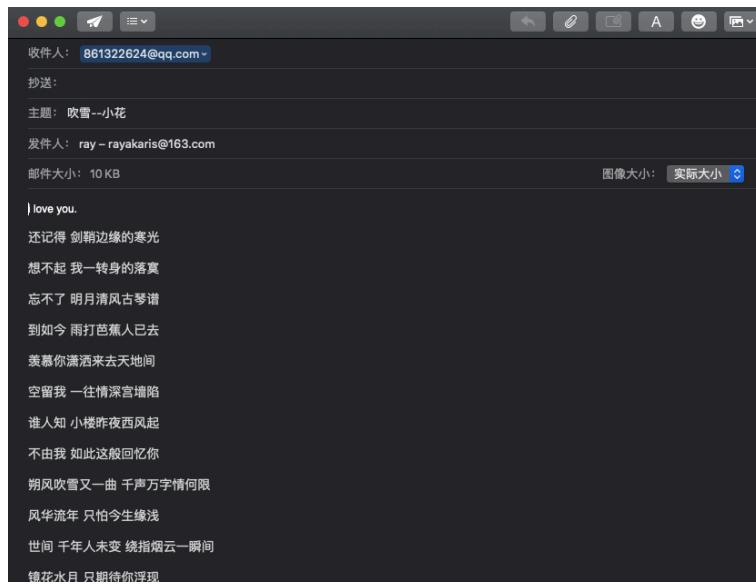
4 实验结果

4.1 初始界面

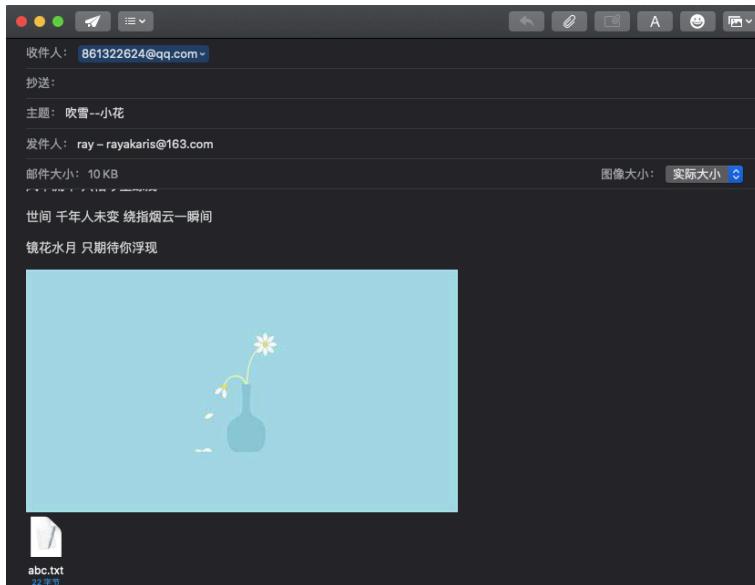


4.2 发送邮件

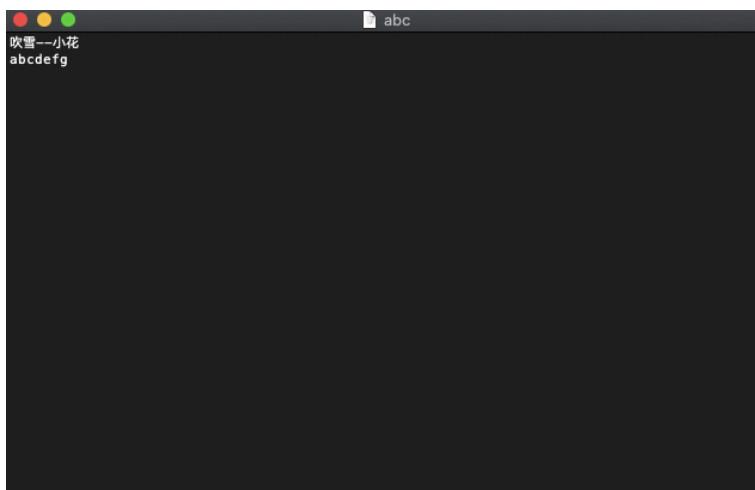
邮件内容如下。



添加附件 (图片和文件)。



附件的文件内容如下。



4.3 通信过程日志及接收到的邮件内容

显示内容如下。



4.4 带格式邮件内容显示

发送的文本内容是有格式的。

1) 纯文本显示



2) 带格式显示

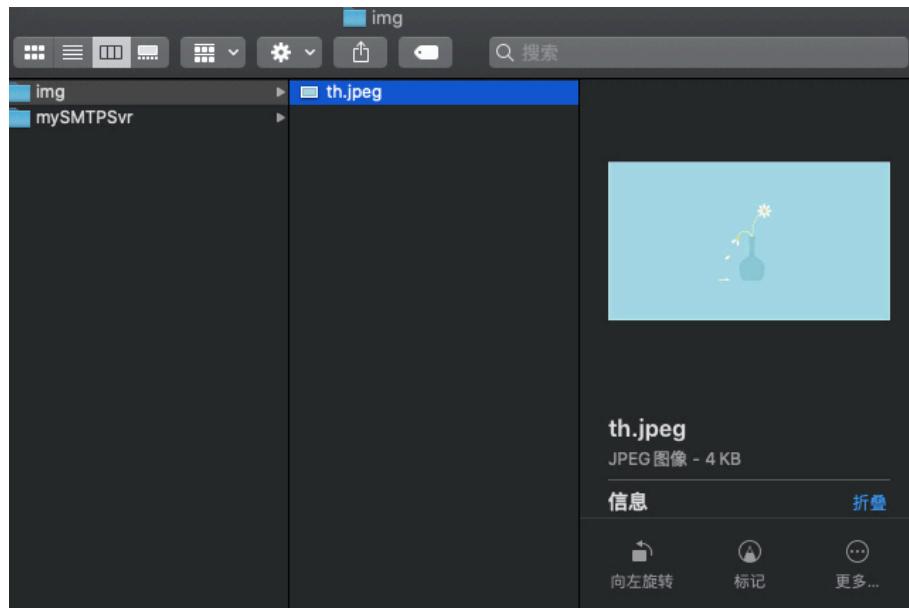


4.5 附件显示及存储

图片附件显示在左下区域，文件附件内容显示如右边部分内容。



附件储存如下。



5 实验总结

5.1 SMTP 邮件的传递过程

- 1) 连接建立阶段：TCP 套接字建立连接；
- 2) 邮件传递阶段：客户端与服务器进行会话、传输数据；
- 3) 连接关闭阶段：会话结束，关闭连接。

5.2 电子邮件报文的组成

- 1) 邮件头：邮件报文的控制信息，由多行组成，每行的基本语法：`<key-word>: <value>`
- 2) 邮件体：用户需要发送的邮件内容，由 7 位 ASCII 文本组成
- 3) 邮件头和邮件体之间用空行分隔

5.3 多用 Internet 邮件扩展协议 MIME

- 1) MIME 解决的主要问题：多媒体等二进制信息能够利用电子邮件传输
- 2) MIME 的基本思想：扩充 RFC822

- 3) MIME 继承了 RFC822 的基本邮件头和邮件体模式
- 4) 增加了一些邮件头字段，要求对邮件体进行编码（将 8 位的二进制信息转换成 7 位的 ASCII 文本）

5.4 基数 64 编码 (base64) 编码算法

- 1) 基本思想：将 3 个字节转换成 4 个可打印字符
- 2) 剩 1 个字节：后面补 4 个比特的“0”，再分成 2 个 6 位组，映射为 2 个 ASCII 字符，而后再填充两个“=”
- 3) 剩 2 个字节：后面补 2 个比特的“0”，再分成 3 个 6 位组，映射为 3 个 ASCII 字符，而后再填充 1 个“=”
- 4) 添加回车换行：变换后，每 76 个字符后增加一回车换行