

实验报告

课程名称 : 计算机网络原理
实验题目 : SMTP客户端编程实验
学号 : 21281280
姓名 : 柯劲帆
班级 : 物联网2101班
指导老师 : 常晓琳
报告日期 : 2024年3月29日

目录

目录

1. 实验目的
2. 实验环境
3. 实验原理
 - 3.1. SMTP传输架构
 - 3.2. SMTP发送原理
 - 3.3. IMAP获取服务器邮件列表原理
4. 实验过程
 - 4.1. 发送邮件
 - 4.1.1. 编写代码
 - 4.1.2. 运行实验
 - 4.2. 获取邮件列表
 - 4.2.1. 编写代码
 - 4.2.2. 运行实验
5. 总结和感想
6. 附录

1. 实验目的

本实验旨在运用各种编程语言实现基于 smtp 协议的 Email 客户端软件。能够对网络编程有进一步的理解和掌握,并能够理解 smtp 协议的细节。

1. 选择合适的编程语言编程实现基于 smtp 协议的 Email 客户端软件。
2. 安装 Email 服务器或选择已有的 Email 服务器,验证自己的 Email 客户端软件是否能进行正常的 Email 收发功能。

2. 实验环境

- **Server OS:** WSL2 Ubuntu-22.04 (Kernel: 5.15.146.1-microsoft-standard-WSL2)
- **邮件服务商:** QQ邮箱

- Python: version 3.11.5

3. 实验原理

3.1. STMP传输架构

用户与用户代理 (user agent) 打交道, 启动用户代理, 键入主题 (subject) 及报文的正文等。在一行上键入一个句点结束报文。用TCP进行的邮件交换是由报文传送代理MTA (Message Transfer Agent) 完成的, 在本实验中我使用的是QQ邮箱作为MTA。用户代理把邮件传给MTA, 由MTA进行交付。

3.2. STMP发送原理

最小SMTP实现支持8种命令。

- **HELO**: 标识自己。参数必须是完全合格的客户主机名。
- **MAIL**: 标识出报文的发起人。
- **RCPT**: 标识接收方。如果有多个接收方, 可以发多个RCPT命令。
- **DATA**: 发送邮件报文的内容。报文的末尾由客户指定, 是只有一个句点的一行。
- **QUIT**: 结束邮件的交换。
- **RSET**: 异常中止当前的邮件事务并使两端复位。丢掉所有有关发送方、接收方或邮件的存储信息。
- **VRIFY**: 使客户能够询问发送方以验证接收方地址, 而无需向接收方发送邮件。通常是系统管理员在查找邮件交付差错时手工使用的。
- **NOOP**: 除了强迫服务器响应一个OK应答码 (200) 外, 不做任何事情。

在发送邮件时需要依次:

1. 发送 **HELO** 命令标识自己;
2. 发送 **MAIL** 标识报文发起人;
3. 发送 **RCPT** 标识接收方;
4. 发送 **DATA**, 含有邮件内容;
5. 若以上任意步骤出现问题, 发送 **RSET** 终止事务并复位;
6. 结束发送使用 **QUIT**。

邮件内容的格式样例如下:

```
1 Content-Type: text/plain; charset="utf-8"
2 MIME-Version: 1.0
3 Content-Transfer-Encoding: base64
4 From: jingfan.ke@qq.com
5 To: jingfan.ke@qq.com
6 Subject: =?utf-8?b?U01UUCDpgq7ku7bmtYvor5U=?=
7
8 6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu277yM5Y+R6YCB6IeqUH10aG9u56iL5bqP44CC
```

内容采用MIME (Multipurpose Internet Mail Extensions) 标准:

- `Content-Type: text/plain; charset="utf-8"`: 指定邮件正文的内容类型为纯文本 (`text/plain`) , 并使用 UTF-8 编码。
- `MIME-Version: 1.0`: 指明这封邮件使用的是MIME版本1.0标准。
- `Content-Transfer-Encoding: base64`: 表明邮件正文的传输编码方式是Base64。
- `From: jingfan.ke@qq.com`: 邮件的发件人地址。
- `To: jingfan.ke@qq.com`: 显示邮件的收件人地址。
- `Subject: =?utf-8?b?U01UUCDpgq7ku7bmtYvor5U=?=`: 邮件主题, 已被 Base64 方式编码。
- 空行: 分隔首部与正文。
- `6L+z5piv5LiA5bCB5rWL6K+v6YKu5Lu277yM5Y+R6YCB6IeqUH10aG9u56iL5bqP44CC`: 邮件正文, 已被 Base64 方式编码。

3.3. IMAP获取服务器邮件列表原理

IMAP (Internet Message Access Protocol) 是一种邮件获取协议, 它允许邮件客户端访问并操作远程邮件服务器上的消息。与仅允许下载的POP (Post Office Protocol) 不同, IMAP提供了更复杂的邮件管理功能, 例如在服务器上保持邮件状态 (已读、未读、删除等), 以及支持在多个客户端之间同步邮件。

通过IMAP获取服务器邮件列表的步骤:

1. **建立连接并认证**: 首先, 客户端使用IMAP协议通过SSL加密的方式连接到邮件服务器的IMAP服务, 并使用用户名和密码进行认证, 确保通信过程的安全性和用户身份的验证。
2. **选择邮件文件夹**: 认证成功后, 客户端选择要操作的邮件文件夹, 通常是“收件箱”。IMAP允许操作多个邮件文件夹, 包括用户自定义的文件夹。
3. **搜索邮件**: 客户端可以根据需要搜索邮件。可以使用 'ALL' 参数来搜索所有邮件。IMAP支持多种搜索标准, 如日期、发件人、主题等, 允许灵活地获取邮件列表。
4. **获取邮件**: 搜索完成后, 服务器返回邮件的唯一标识符 (ID) 。然后, 客户端可以根据这些ID获取一封或多封邮件的完整内容或部分内容。可以通过遍历邮件ID列表, 并使用 `fetch` 命令按照RFC 822标准获取邮件的完整数据。
5. **解析邮件内容**: 邮件内容通常以MIME格式存储, 包含多部分内容 (如文本、HTML、附件等) 。客户端需要解析这些内容, 提取邮件的主体、主题、发件人等信息。可以使用 `email` 模块来解析邮件内容, 并处理多部分消息和文本编码。
6. **处理邮件**: 获取并解析邮件后, 可以根据需要处理邮件, 例如显示邮件列表、保存邮件到本地或对邮件进行标记处理。
7. **断开连接**: 操作完成后, 客户端发送 `logout` 命令来结束会话, 并断开与服务器的连接。

通过以上步骤, IMAP协议支持的邮件客户端能够高效地管理和操作服务器上的邮件, 支持复杂的邮件处理需求, 特别适用于需要在多个设备上访问和同步邮件的场景。

4. 实验过程

4.1. 发送邮件

4.1.1. 编写代码

首先写一个配置文件 `data/email_config.json`，指明发件人、收件人、邮件主题、邮件正文、SMTP服务器和密码：

```
1 {
2     "sender": "jingfan.ke@qq.com",
3     "receiver": "jingfan.ke@qq.com",
4     "subject": "SMTP 邮件测试",
5     "body": "这是一封测试邮件，发送自Python程序。",
6     "smtp_server": "smtp.qq.com",
7     "password": "xxxxxxxxxxxxxx"
8 }
```

该密码通过开通QQ邮箱的SMTP/POP3/IMAP服务获得。

在代码文件 `sender.python` 中：

1. 从配置文件中提取所需的信息，包括发件人、收件人、邮件主题、邮件正文、SMTP服务器和密码；

```
1 with open('./data/email_config.json', 'r') as config_file:
2     config = json.load(config_file)
3
4 sender = config['sender']
5 receiver = config['receiver']
6 subject = config['subject']
7 body = config['body']
8 smtp_server = config['smtp_server']
9 password = config['password']
```

2. 创建一个 `MIMEText` 对象，用于设置邮件的正文。邮件内容设为纯文本格式 ('plain')，字符集为 'utf-8'。

```
1 message = MIMEText(body, 'plain', 'utf-8')
```

3. 设置邮件头部信息，包括发件人、收件人和邮件主题。使用 `Header` 来确保头部信息能够正确处理字符编码。

```
1 message['From'] = Header(sender)
2 message['To'] = Header(receiver)
3 message['Subject'] = Header(subject, 'utf-8')
```

4. 尝试执行以下步骤来连接SMTP服务器并发送邮件：

- 使用 `SMTP_SSL` 连接到指定的SMTP服务器和端口（这里使用465端口，它通常用于SMTPS即加密SMTP）。

```
1 | server = smtplib.SMTP_SSL(smtp_server, 465)
```

- 使用提供的发件人邮箱地址和密码登录SMTP服务器。

```
1 | server.login(sender, password)
```

- 调用 `sendmail` 方法发送邮件。这里将邮件内容转换为字符串格式，并指定发件人和收件人地址。

```
1 | server.sendmail(sender, [receiver], message.as_string())
```

`sendmail` 方法在内部依次发送了 `HELO` 命令、`MAIL` 命令、`RCPT` 命令和 `DATA` 及内容，这部分将在运行实验部分详细叙述。

- 如果邮件发送成功，则打印“邮件发送成功”的消息。
5. 如果在邮件发送过程中遇到任何 `SMTPException` 异常，则捕获这个异常并打印“邮件发送失败”的消息，同时显示错误详情。
 6. 不论邮件发送成功与否，最后都会调用 `quit` 方法来关闭与SMTP服务器的连接。

```
1 | server.quit()
```

以上完整代码见附录。

4.1.2. 运行实验

将代码文件放在合适的位置，文件夹构建如下：


```
1 | .
2 | └─ data
3 |   └─ email_config.json
4 | └─ sender.py
```

使用Python运行代码：

```
1 | $ python sender.py
2 | 邮件发送成功
```

发送目标邮箱即可收到邮件。

SMTP 邮件测试 ☆

发件人: 柯劲帆 <jingfan.ke@qq.com> 
时 间: 2024年3月29日 (星期五) 上午11:03
收件人: 柯劲帆 <jingfan.ke@qq.com>

这是一封测试邮件，发送自Python程序。

那么在发送邮件的过程中，究竟发送了什么具体的内容呢？

首先是邮件内容 `message` 字符串，打印内容为：

```
1 Content-Type: text/plain; charset="utf-8"
2 MIME-Version: 1.0
3 Content-Transfer-Encoding: base64
4 From: jingfan.ke@qq.com
5 To: jingfan.ke@qq.com
6 Subject: =?utf-8?b?U01UUCDpgq7ku7bmtYvor5U=?=
7
8 6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu277yM5Y+R6YCB6IeqUH10ag9u56iL5bqP44CC
```

经过对函数源码的深入解读，我发现 `sendmail()` 方法内部实现了发送邮件的主要逻辑：

1. 调用了 `ehlo_or_helo_if_needed()` 方法发送了 `HELO` 命令：

函数调用栈：`ehlo_or_helo_if_needed()` → `helo()` → `putcmd()` → `send()` → `sock.sendall()`，在 `sock.sendall()` 后添加打印语句 `print()`，打印socket发送的字符串为：

```
1 | b'ehlo kkkkjf.\r\n'
```

2. 调用了 `mail()` 方法发送了 `MAIL` 命令：

函数调用栈：`mail()` → `putcmd()` → `send()` → `sock.sendall()`，打印socket发送的字符串为：

```
1 | b'mail FROM:<jingfan.ke@qq.com> size=264\r\n'
```

3. 对每个发送对象，调用一次 `rcpt()` 方法发送 `RCPT` 命令：

函数调用栈：`rcpt()` → `putcmd()` → `send()` → `sock.sendall()`，打印socket发送的字符串为：

```
1 | b'rcpt TO:<jingfan.ke@qq.com>\r\n'
```

这里由于只有一个发送对象，所以只有一行。

4. 调用了 `data()` 方法发送了 `DATA` 命令：

函数调用栈：

1. `data()` → `putcmd()` → `send()` → `sock.sendall()`，打印socket发送的字符串为：

```
1 | b'data\r\n'
```

2. `data()` → `send()` → `sock.sendall()`，打印socket发送的字符串为：

```
1 | b'Content-Type: text/plain; charset="utf-8"\r\nMIME-Version:
  1.0\r\nContent-Transfer-Encoding: base64\r\nFrom:
  jingfan.ke@qq.com\r\nTo: jingfan.ke@qq.com\r\nSubject: =?utf-8?b?
  U01UUCDpgq7ku7bmtYvor5U=?
  =\r\n\r\n6L+Z5piv5LiA5bCB5rWL6K+V6YKu5Lu277yM5Y+R6YCB6IeqUH10ag9u56iL
  5bqP44CC\r\n.\r\n'
```

这正是 `message` 字符串内容。

在打印 邮件发送成功后，调用了 `server.quit()`，探究源码后发现执行逻辑为：

函数调用栈: `rcpt()` → `docmd()` → `putcmd()` → `send()` → `sock.sendall()`, 打印socket发送的字符串为:

```
1 | b'quit\r\n'
```

至此, 邮件发送完毕。

4.2. 获取邮件列表

4.2.1. 编写代码

首先需要在 `data/email_config.json` 中添加IMAP服务器的地址。

```
1 | {
2 |     "sender": "jingfan.ke@qq.com",
3 |     ...
4 |     "imap_url": "imap.qq.com",
5 |     "password": "xxxxxxxxxxxxxxx"
6 | }
```

接下来在代码文件 `receiver.python` 中:

1. 配置加载

从JSON文件中加载邮件客户端配置信息。使用 `json.load` 方法读取文件 `data/email_config.json` 中的配置信息, 包括用户的邮箱地址、密码、以及IMAP服务器的URL, 并将文件内容解析为Python字典。

2. 连接和登录

使用 `imaplib.IMAP4_SSL()` 方法建立与IMAP服务器的安全连接, 然后使用 `login()` 方法进行用户认证。保证后续操作的安全性和用户身份的验证。

3. 邮件搜索和获取

在成功登录后, 客户端选择“收件箱”文件夹, 并搜索所有邮件。通过 `search()` 方法获取到的邮件ID列表, 用于后续获取邮件的详细内容。

4. 邮件解析

遍历邮件ID列表, 使用 `fetch()` 方法获取每封邮件的完整数据 (RFC822)。接着使用 `email.message_from_string()` 方法解析这些数据, 提取邮件的发件人、主题和正文等信息。这一部分考虑到邮件可能是多部分格式, 分别处理了邮件正文是单一部分和多部分的情况。

5. 邮件信息保存

将解析出的邮件信息保存到一个新的JSON文件中。使用 `json.dump()` 方法, 将邮件数据写入文件, 以便后续的查阅或分析。

6. 输出和断开连接

最后, 打印最后一封邮件的发件人、主题和正文信息作为示例输出, 并使用 `logout()` 方法断开与IMAP服务器的连接。

以上完整代码见附录。

4.2.2. 运行实验

代码文件和即将保存的邮件文件在文件夹中的位置：

```
1 | .
2 | └─ data
3 |   └─ emails.json
4 |   └─ receiver.py
```

运行代码：

```
1 | $ python receiver.py
2 | From: jingfan.ke@qq.com
3 | Subject: SMTP 邮件测试
4 | Body: 这是一封测试邮件，发送自Python程序。
```

打印了最后一封邮件。

所有邮件列表保存在文件 `data/emails.json` 中，截图如下：



```
data > {} emails.json > {} 2 > Body
1 | [
2 |   {
3 |     "From": "jingfan.ke@qq.com",
4 |     "Subject": "SMTP 邮件测试",
5 |     "Body": "这是一封测试邮件，发送自Python程序。"
6 |   },
7 |   {
8 |     "From": "jingfan.ke@qq.com",
9 |     "Subject": "SMTP 邮件测试",
10 |    "Body": "这是一封测试邮件，发送自Python程序。"
11 |   },
12 |   {
13 |     "From": "jingfan.ke@qq.com"
```

5. 总结和感想

通过本次实验，我深入学习和理解了SMTP协议的工作原理，实际操作了如何使用Python编程语言实现基于SMTP协议的Email客户端软件。实验中，我成功完成了邮件的发送和接收功能，对于网络编程有了更加深刻的认识。

在实验的过程中，我遇到了一些挑战，比如理解SMTP和IMAP协议的细节、邮件内容的格式化、以及如何使用Python的 `smtp` 和 `imaplib` 库来实现邮件的发送和接收。通过查询官方文档和一些技术博客，我逐步克服了这些难题，对这些协议的理解也更加深入了。

此外，我还学习到了如何使用 `json` 文件来管理配置信息，这种方法不仅使代码更加清晰，也让程序的可维护性和可扩展性大大提高。通过实验，我发现了编码实践对于加深理论知识的理解有着不可替代的作用。

总的来说，这次实验不仅让我对网络编程有了更进一步的了解，而且还提高了我解决实际问题的能力。通过动手实践，我学习到了许多课本之外的知识，这将对我未来的学习和研究工作大有裨益。我期待在未来能够继续探索更多关于计算机网络以及其他计算机科学领域的知识和技术。

6. 附录

sender.py:

```
1 import smtplib
2 import json
3 from email.mime.text import MIMEText
4 from email.header import Header
5
6 # 从JSON文件中加载邮件配置
7 with open('./data/email_config.json', 'r') as config_file:
8     config = json.load(config_file)
9
10 sender = config['sender']
11 receiver = config['receiver']
12 subject = config['subject']
13 body = config['body']
14 smtp_server = config['smtp_server']
15 password = config['password']
16
17 # 创建MIMEText对象，设置邮件内容
18 message = MIMEText(body, 'plain', 'utf-8')
19 message['From'] = Header(sender)
20 message['To'] = Header(receiver)
21 message['Subject'] = Header(subject, 'utf-8')
22
23 try:
24     # 连接SMTP服务器，并发送邮件
25     server = smtplib.SMTP_SSL(smtp_server, 465) # 使用465端口
26     server.login(sender, password)
27     server.sendmail(sender, [receiver], message.as_string())
28     print("邮件发送成功")
29     server.quit()
30 except smtplib.SMTPException as e:
31     print("邮件发送失败", e)
```

receiver.py:

```
1 import imaplib
2 import email
3 import json
4 from email.header import decode_header
5
6 # 从JSON文件中加载配置信息
7 with open('./data/email_config.json', 'r') as config_file:
8     config = json.load(config_file)
9
10 user = config['sender']
```

```

11 password = config['password']
12 imap_url = config['imap_url']
13
14 # 连接到IMAP服务器
15 mail = imaplib.IMAP4_SSL(imap_url)
16 mail.login(user, password)
17 mail.select('inbox')
18
19 # 搜索所有邮件
20 result, data = mail.search(None, 'ALL')
21 mail_ids = data[0]
22
23 id_list = mail_ids.split()
24 id_list.reverse()
25 emails = []
26
27 # 遍历邮件ID
28 for i in id_list:
29     result, data = mail.fetch(i, '(RFC822)')
30     raw_email = data[0][1]
31     raw_email_string = raw_email.decode('utf-8', errors="ignore")
32     email_message = email.message_from_string(raw_email_string)
33
34     # 解析邮件内容
35     mail_from = email_message['From']
36     mail_subject = decode_header(email_message['Subject'])[0][0]
37     if isinstance(mail_subject, bytes):
38         mail_subject = mail_subject.decode('utf-8')
39     mail_body = ''
40     if email_message.is_multipart():
41         for part in email_message.walk():
42             ctype = part.get_content_type()
43             cdispo = str(part.get('Content-Disposition'))
44             if ctype == 'text/plain' and 'attachment' not in cdispo:
45                 mail_body = part.get_payload(decode=True).decode('utf-8')
46                 break
47     else:
48         mail_body = email_message.get_payload(decode=True).decode('utf-8')
49
50     emails.append({'From': mail_from, 'Subject': mail_subject, 'Body':
51 mail_body})
52
53 # 保存邮件列表到JSON文件
54 with open('./data/emails.json', 'w') as outfile:
55     json.dump(emails, outfile, indent=4, ensure_ascii=False)
56
57 # 打印最后一封邮件的信息
58 if emails:
59     print("From: ", emails[0]['From'])
60     print("Subject: ", emails[0]['Subject'])
61     print("Body: ", emails[0]['Body'])
62
63 mail.logout()

```

