

基于电子邮件方式传输数据的软件模块设计

付晓光, 汪秉文, 王文顺

(华中科技大学 控制科学与工程系, 湖北 武汉 430074)

摘 要: 研究并设计了电子邮件 MUA(Mail User Agent, 邮件用户代理) 软件模块, 给出了实现电子邮件数据流处理及通讯协议实现的具体细节。该模块严格遵循 RFC 文档, 融合服务器认证过程, 可以接入 Internet 上规范的邮件服务器; 提供丰富的程序接口, 可作为应用系统中的一个通讯模块使用。

关键词: 电子邮件; 简单邮件传输协议; 邮件用户代理; NTLM

中图分类号: TP393.098

文献标识码: A

文章编号: 1673-629X(2007)04-0235-04

Design of a Software Module for Data Exchange Based on Email

FU Xiao-guang, WANG Bing-wen, WANG Wen-shun

(Control Science and Engineering Department, Huazhong University of
Science and Technology, Wuhan 430074, China)

Abstract: A MUA (Mail User Agent) software module is researched and designed. To give more detail, it presents how the Email data stream is processed and how the communication protocols are implemented. This module strictly conforms to the RFC document as well as server authenticate process is fixed in it, as a result of which it can connect with normative Internet mail server easily. At the same time, you can use it as a communication module.

Key words: Email; SMTP; MUA; NTLM

0 引言

常见的电子邮件用户代理程序 FOXMAIL 及 OUTLOOK 为人们日常生活收发电子邮件提供了极大的便利。但是这些应用程序属于商用软件体系, 并不能够提供应用于系统开发的程序接口。因此, 如果系统需要使用电子邮件方式传递数据, 就只能自主开发 MUA(Mail User Agent, 邮件用户代理) 模块。这个软件模块具有两大特点: 一是具有可扩展程序接口、应用灵活; 二是严格遵循 RFC 文档中定义的内容, 并加入服务器认证过程, 可以接入 Internet 上规范的邮件服务器。整个软件模块从最低层的 SOCKET 接口层开始, 考虑并且支持 SOCKS/HTTP 代理协议, 从而可以保证开放性及健壮性。

1 总体设计

在 Internet 上, 电子邮件的整个传输过程如图 1 所示。

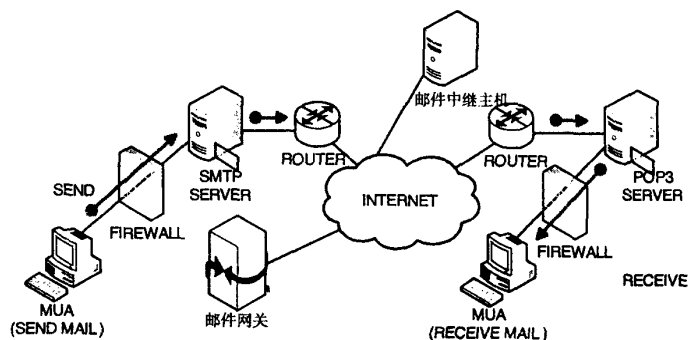


图 1 电子邮件网络传输示意图

依循数据处理和协议实现分开的思路, 这个 MUA 的子模块有如下几个: EMAIL 信件标准数据流处理、SOCKET 接口层、邮件服务器认证、SMTP 信件发送过程和 POP3 信件接收过程。

该模块是在 Windows + MFC 开发平台下实现的, 因此笔者以此平台为例分别就几个子模块进行详细描述。

收稿日期: 2006-06-19

作者简介: 付晓光(1977-), 男, 黑龙江哈尔滨人, 硕士研究生, 研究方向为计算机集成控制与网络技术; 汪秉文, 教授, 博士生导师, 研究方向为生产过程综合自动化集成优化与决策, 计算机网络与数据库系统。

2 EMAIL 信件标准数据流处理

网络上传输的 EMAIL 数据流是一些纯粹的 ASCII 码并且是可打印的字符(不包括 CR 和 LF),使用 ASCII 码(7bit)进行数据传输是 EMAIL 服务的特点。这个子模块的功能就是在发送时把邮件编码处理成标准的 EMAIL 数据流,在接收时解码数据流还原邮件,最合适的数据流定义就是 C++ 标准语法:

```
std::string
```

一般采用形如“李平<LiPing@sina.com>”来表示一个完整描述的信箱。“李平”在 MUA 发送方系统里可能是以 UTF-8(UNICODE)、代码页(cp936)或 UCS-2(UNICODE)来表示,这样它的内码表示中某个字节的数据肯定将超出 7bit,这种非 ASCII 码的字符一定要经过“Q encoding”^[1]的过程进行编码处理。处理后的信息变成 ASCII 字符串并包含了原始的字符集信息。有了这些信息,MUA 接收端就可以完全将其还原。

一封邮件在结构上可以分为信头和信体两个部分,程序实现上就是要定义一个邮件消息类,信头是以 CString 类型定义,信体处理部分定义一个类来实现。数据流中信体与信头之间以“CRLF CRLF”分界^[1,2]。不论接收或发送,信体和信头都是单独处理的,因此需要两个函数:

```
Class CMAILMessage { ...
```

```
std::string getHeader();//生成信头数据流
```

```
std::string getBody();//生成信体数据流
```

```
CMAILAddress m_ From;//发信人
```

```
CArray< CMAILAddress *, CMAILAddress * &> m_ ToRecipients;//收信人
```

```
CString m_ sSubject; //主题
```

```
CString m_ sXMailer; //发信客户端
```

```
CMAILMSGBodyPart m_ RootPart;//根信体
```

```
...};
```

信头中只考虑那些必要的域,如“From:”,“To:”,“Subject:”,“X-Mailer:”,“MIME-Version:”,“Content-Type:”,“Date:”。需要注意两点:一是在使用 MIME 格式的时候,一定要把“MIME-Version: 1.0”这句加到信头中去,否则 MUA 接收端即使支持 MIME,也会以不确定的方式来处理这封信^[1];二就是最好要加上 X-Mailer 信头,没有或者使用特殊的 X-Mailer 信头,可能被一些邮箱作为垃圾邮件而拒收。最后就是为“Content-Type:”域的 boundary 子选项填入一个唯一的 ASCII 长字符串作为多信体时的各信体之间的分界符。

信体允许嵌套,应按如下树型定义:

```
Class CMAILMSGBodyPart
```

```
{ ...
```

```
CMAILMSGBodyPart * m_ pParentBodyPart;//父信体
```

```
CArray< CMAILMSGBodyPart *, CMAILMSGBodyPart * &> m_ ChildBodyParts;//子信体
```

```
CString m_ sAttachFilename;//附件文件名
```

```
CString m_ sPlainText;//明文
```

```
CString m_ sCharset;//明文字符集
```

```
...};
```

对于信体只考虑明文(text/plain)及附件(application/octet-stream)两种。这里直接给出明文信体和附件信体的头描述:

```
Content-Type: text/plain; charset = gb2312
```

```
Content-Transfer-Encoding: quoted-printable
```

```
Content-Type: application/octet-stream; name = "attachFile.dat"
```

```
Content-Transfer-Encoding: base64
```

```
Content-Disposition: attachment; filename = "attachFile.dat"
```

正如这个信体的头部描述,使用 quoted-printable^[1]方式对明文进行编码,使用 BASE64^[1]对附件编码。相应地,在 MUA 收到邮件时再用“Content-Transfer-Encoding:”域指定的方式对其进行解码。另外,明文信体还需要对指定的字符集进行内码转换。

每个邮件消息以仅由单个点号构成的一行结束,改用 ASCII 字符名称来说就是每个邮件消息的信体必须以“CRLF.CRLF”结尾。在这种方式下,当从同一个 MUA 接收一系列邮件消息时,SMTP 服务器可以通过在字节流中搜索“CRLF.CRLF”来分割每个消息。在信体发送时生成的二进制数据字节流中偶尔会出现“CRLF.CRLF”这一模式,这将导致 SMTP 服务器错误地认定当前邮件消息已结束。为避免这样的问题,把不代表消息结束的“CRLF.CRLF”中的一个点变成两个点^[2]。

这个子模块不论是对邮件编码还是解码都要进行语法分析,需要大量的字符串处理代码,用面向对象的方法很容易理解并实现。部分译码、解码的例程在相应的 RFC 中可以找到。

3 协议实现

3.1 SOCKET 接口层

使用 WIN32 SOCKET API 封装 SOCKET 类,实现与 BERKELEY SOCKET 代码兼容,程序更加可靠和灵活。加入对代理协议的支持就可以透过防火墙的限制访问邮件服务器,而代理协议都定义有连接方法,因此对 SOCKS/HTTP PROXY 代理的支持放在这个子模块中。

关于 SOCKS 代理协议,其中 SOCKS4 协议比较简

单,只需要发送一次请求,服务器方就会发出是否可以代理资源的应答^[3]。SOCKS5 协议相对复杂,要与服务器进行多次问答,并与服务器端确定关于目的地址是 IP 地址还是域名及具体的协议细节等等。其中要注意的是,第一次请求中关于 SOCKS5 定义的连接方法集(METHOD SET)^[3]中,只要指定需要认证/不需要认证两种方法。

HTTP PROXY 实际按照 HTTP/1.1 协议^[4]发送具体 URL 请求即可,一个需要注意的问题就是要使用 BASE64 编码发送代理服务器端需要的用户名/密码。

由以上可知,在这部分代码中除了常规的 SOCKET 函数外还要定义如下几个函数:

```
ConnectViaSocks4( ... );
```

```
ConnectViaSocks5( ... );
```

```
ConnectViaHTTPProxy( ... );
```

3.2 服务器认证

在 Internet 中的 MAIL 服务器大部分都需要认证,目前主要有 4 种认证方式:CRAM-MD5 认证、明文认证、登陆认证和 NTLM 认证。

服务器认证涉及到数据加密方面的问题,Windows 编程需要 PLATFORM SDK 的 Wincrypt.h 及相应的链接库,用来计算 HASH 及 HMAC 值。HASH 算法采用比较常见的 MD5 算法,HMAC^[5]就是有密钥的 MD5。

这里主要讨论 NTLM 认证方式。首先调用 GenClientContext 函数生成客户端的环境安全信息(client context),然后开始与服务器的认证交换过程:

(1)把 GenClientContext 输出的信息用 BASE64 编码后发到服务器,开始认证请求。

(2)服务器发回 334 命令码同意认证,同时发回一串用 BASE64 编码的信息作为 CHALLENGE。

(3)客户端取出 CHALLENGE 信息,用 BASE64 解码该信息,然后再次调用 GenClientContext 函数生成客户端的 RESPONSE。

(4)将 RESPONSE 信息发送到服务器端。服务器端将 CHALLENGE 及 RESPONSE 发送到域控制器,判断是否通过认证。

把(2)~(4)步骤的代码放到一个循环体内,直到认证成功退出。

3.3 SMTP 协议实现

在建立连接时,可以根据选定的 PROXY 调用相关的 SOCKET 函数。对于需要认证的 SMTP 服务器应该使用扩展 SMTP 协议,连接后的第一个命令是“EHLO”^[6],而常规 SMTP 是“HELO”^[7]。图 2 给出实现这个协议的流程(注意每个命令都要以 CRLF 结

束)。

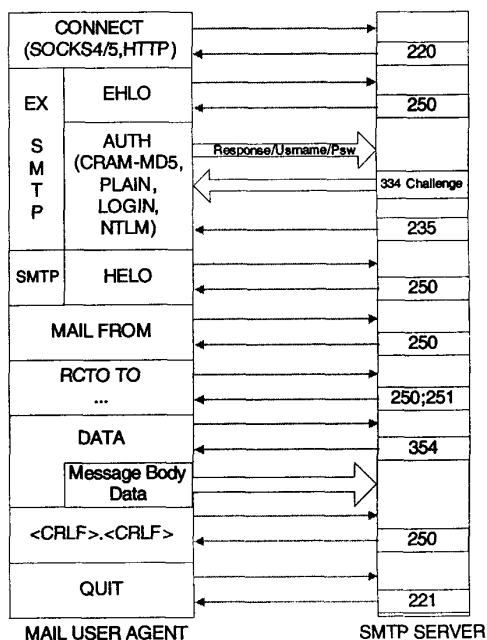


图 2 SMTP 协议命令与应答流程

由于在形成邮件数据流时是按照树型结构定义,所以使用一个递归函数来实现信体(BODY)的发送,并根据每个信体是否拥有子信体来控制递归深度。

这部分还要实现邮件的路由问题。假设要把邮件从 LiPing@sina.com 发送到 Tom@163.com,可以直接把信投到 smtp.sina.com.cn 这个服务器上,这个服务器必然会直接或者间接地把信投到目的邮箱。还有一种情况就是不知道该投到那个服务器,这时候就需要查询 DNS 来获得想要的服务器地址。如上例对信箱的域名即 163.COM 进行 DNS 查询。注意我们关心的只是邮件服务器的 MX 记录, MX 记录就是这个邮件服务器和别的邮件服务器的邮件交换记录,通过 MX 记录可以查到多个直接或间接与目的邮件服务器相关的记录,并给出投递的优先级,只取优先级最高的邮件服务器,把邮件投过去即可。这个部分功能通过调用 DnsQuery 函数实现,注意要指定只查询 MX 记录。

针对 SMTP 协议应用最小流程封装一个 CSMTP-Connect 类。主要成员如下:

```
Class CSMTPConnect { ...
void AuthNTLM(...); //服务器认证
void SendRCPTForRecipient(...); //发送 RCPT 命令
void SendBodyPart(...); //信体发送
BOOL ReadResponse(...); //读取服务器响应,要做语法分析
static BOOL MXLookup(...); //MX 记录查询
CWSocket m_Socket; //SOCKET 连接
int m_nLastCommandResponseCode; //服务器最新的命令码
```

```
CString m_sProxyServer;//代理服务器 IP 或域名
int m_nProxyPort;//代理服务器端口
...};
```

3.4 POP3 协议实现

POP3 响应由一个状态码和一个可能跟有的附加信息组成,所有响应也是以 CRLF 对结束。POP3 响应共有两种状态码“+OK”和“-ERR”,其对特定命令的响应是由许多字符组成的。在发送第一行响应和一个 CRLF 之后,任何的附加信息行也以 CRLF 对结束,发送的最后一行信息应包括一个结束字符“.”和一个 CRLF 对。检测多行响应时,客户应确认此行是否以结束字符开始,如果是而且其后的字符不是 CRLF,此行的第一个结束字符应被抛弃[8]。

POP3 的协议实现不像 SMTP 一样有清晰的流程。它有认证、传输和更新三种状态,每种状态下只有相应的命令才可以提交 POP3 服务器并作相应状态转换,对于在模

块中实际用到的命令如表 1 所示。

4 结束语

介绍了一个基于 Windows 平台的 MUA 软件模块的实现过程。该模块已成功在长庆油田采油二厂原油集输站数据采集系统上应用,在这里给出一个该 MUA 模块发送端的测试用界面(如图 3 所示)。

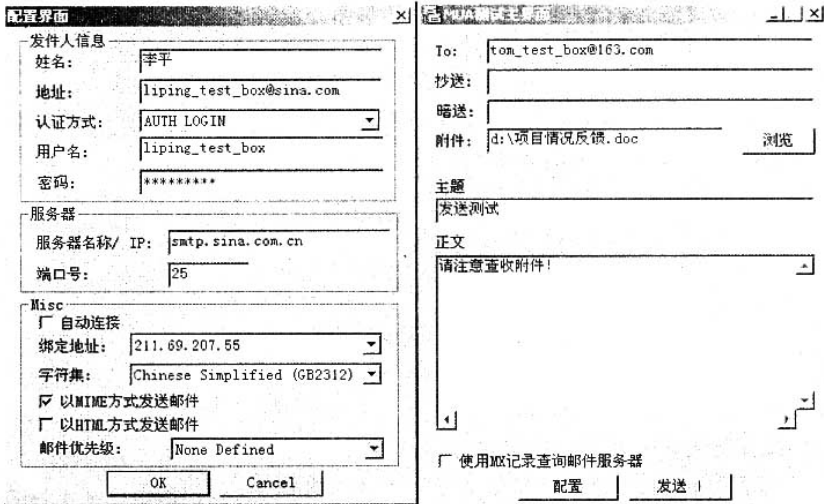


图 3 测试界面

经运行测试,这个模块具有很好的通用性,适用于现有的大部分公用邮箱。如果系统对邮件安全性有较高的要求,可以考虑加入对 OPENSSL 的支持。

参考文献:

[1] RFC2045~2049. Multipurpose Internet Mail Extensions (MIME)[S]. 1996.
[2] RFC822. Standard for the format of ARPA Internet text messages[S]. 1982.
[3] RFC1928. Socks Protocol Version 5[S]. 1996.
[4] RFC2616. HyperText Transfer Protocol - HTTP/1.1[S]. 1999.
[5] RFC2104. HMAC: Keyed - Hashing for Message Authentication[S]. 1997.
[6] RFC1869. SMTP Service Extensions[S]. 1995.
[7] RFC821. Simple Mail Transfer Protocol[S]. 1982.
[8] RFC1939. Post Office Protocol version - 3[S]. 1996.

表 1 POP3 协议命令与应答流程

状态	命令	应答	描述
认证 Authentication	Socket 连接	+ OK POP3 server ready	打开连接
	USER name	+ OK POP3	用户名
	PASS string	+ OK POP3	密码。如认证成功转入传输状态
	QUIT	+ OK POP3 Disconnect	此次 QUIT 命令不进入更新状态
传输 Transaction	STAT	+ OK 2 320	统计
	LIST [msg id]	+ OK 2 messages (320 octets) 1 120 2 200	统计列表
	RETR msg id	+ OK 120 octets < 服务器发送信件 >	收邮件
	DELE msg id	+ OK message 1 deleted	加删除标记
	NOOP	+ OK	空闲
	RSET	+ OK	服务器去掉某些邮件的删除标记
	QUIT	+ OK Server signing off	进入更新状态

(上接第 234 页)

[6] 李 凡,饶 勇. 基于 Vague 集的加权模糊运算[J]. 华中科技大学学报:自然科学版,2001,29(3):25-29.
[7] Kumar D S, Biswas R, Ranjan R A. An application of intuitionistic fuzzy sets in medical diagnosis[J]. Fuzzy Sets and Systems, 2001,117(2):209-213.

[8] Chen S M. Fuzzy system reliability analysis based on vague sets theory[C]//Proceeding of the 1997 IEEE International Conference on Computational Cybernetics and Simulation. Orlando: IEEE Press, 1997:1650-1655.