一、设计要求

实现 FTP 客户端，能列出/更改目录，上下传文件，显示文件列表等选项。

二、开发环境与工具

开发环境：Windows 10

开发工具：Visual Studio 2019

开发语言：C#

开发平台：Winform .Net Framework 4.7.2

测试工具：

1. 虚拟机软件：VMware WorkStation 15 Player
2. 虚拟机操作系统：Ubuntu 18.04 64 位
3. FTP 服务器软件：vsftpd-3.0.3
4. 抓包软件：Wireshark

三、设计原理

(一) FTP 简介[1]

FTP(File Transfer Protocol)是应用层的一个文件传输协议。其主要作用是在服务器和客户端之间实现文件的传输和共享。FTP 协议运行在 TCP 连接上，保证了文件传输的可靠性。

(二) FTP 的两种传输方式

FTP 支持两种方式的传输：文本（ASCII）方式和二进制（Binary）方式。通常文本文件的传输采用 ASCII 方式，而图像、声音文件、加密和压缩文件等非文本文件采用二进制方式传输。FTP 以 ASCII 方式作为默认的文件传输方式。

(三) FTP 的两种传输模式[2]

FTP 有两种传输模式：主动（FTP Port）模式和被动（FTP Passive）模式。由于主动模式存在着安全问题，最近几年，大部分的 FTP 客户端开始默认使用被动模式。

1. 主动模式（Port）

(1) 客户端发送一个 TCP SYN（TCP 同步）包给服务器段众所周知的 FTP 控制端口 21，客户端使用暂时的端口作为它的源端口；

(2) 服务器端发送 SYN ACK（同步确认）包给客户端，源端口为 21，目的端口为客户端上使用的暂时端口；

(3) 客户端发送一个 ACK（确认）包；客户端使用这个连接来发送 FTP 命令，服务器端使用这个连接来发送 FTP 应答；

(4) 当用户请求一个列表(List)请求或者发起一个要求发送或者接受文件的请求，客户端软件使用 PORT 命令，这个命令包含了一个暂时的端口，客户端希望服务器在打开一个数据连接时候使用这个暂时端口；PORT 命令也包含了一个 IP 地址，这个 IP 地址通常是客户自己的 IP 地址，而且 FTP 也支持第三方（third-party）模式，第三方模式是客户端告诉服务器端打开与另台主机的连接；

(5) 服务器端发送一个 SYN 包给客户端的暂时端口，源端口为 20，暂时端口为客户端在 PORT 命令中发送给服务器端的暂时端口号；

(6) 客户端以源端口为暂时端口，目的端口为 20 发送一个 SYN ACK 包；

(7) 服务器端发送一个 ACK 包；

(8) 发送数据的主机以这个连接来发送数据，数据以 TCP 段(注：segment，第 4 层的 PDU)形式发送（一些命令，如 STOR 表示客户端要发送数据，RETR 表示服务器段发送数据），这些 TCP 段都需要对方进行 ACK 确认（注：因为 TCP 协议是一个面向连接的协议）

(9) 当数据传输完成以后，发送数据的主机以一个 FIN 命令来结束数据连接，这个 FIN 命令需要另一台主机以 ACK 确认，另一台主机也发送一个 FIN 命令，这个 FIN 命令同样需要发送数据的主机以 ACK 确认；

(10) 客户端能在控制连接上发送更多的命令，这可以打开和关闭另外的数据连接；有时候客户端结束后，客户端以 FIN 命令来关闭一个控制连接，服务器端以 ACK 包来确认客户端的 FIN，服务器同样也发送它的 FIN，客户端用 ACK 来确认。

2. 被动方式（Passive）

由于主动方式中，服务端需要主动连客户端，对于客户端的防火墙来说，属于外部连接内部，会出现被阻塞的情况。被动方式解决了这个问题。被动连接的核心是控制连接请求和数据连接请求都是由客户端发起。被动方式的步骤如下：

(1) 客户端发送一个 TCP SYN（TCP 同步）包给服务器段众所周知的 FTP 控

制端口 21，客户端使用暂时的端口作为它的源端口；

(2) 服务器端发送 SYN ACK（同步确认）包给客户端，源端口为 21，目的端口为客户端上使用的暂时端口；

(3) 客户端发送一个 ACK（确认）包；客户端使用这个连接来发送 FTP 命令，服务器端使用这个连接来发送 FTP 应答；

(4) 当用户请求一个列表(List)或者发送或接收文件时候，客户端软件发送 PASV 命令给服务器端表明客户端希望进入 Passive 模式；

(5) 服务器端进行应答，应答包括服务器的 IP 地址和一个暂时的端口，这个暂时的端口是客户端在打开数据传输连接时应该使用的端口；

(6) 客户端发送一个 SYN 包，源端口为客户端自己选择的一个暂时端口，目的端口为服务器在 PASV 应答命令中指定的暂时端口号；

(7) 服务器端发送 SYN ACK 包给客户端，目的端口为客户端自己选择的暂时端口，源端口为 PASV 应答中指定的暂时端口号；

(8) 客户端发送一个 ACK 包；

(9) 发送数据的主机以这个连接来发送数据，数据以 TCP 段(注：segment，第 4 层的 PDU)形式发送（一些命令，如 STOR 表示客户端要发送数据，RETR 表示服务器段发送数据），这些 TCP 段都需要对方进行 ACK 确认；

(10) 当数据传输完成以后，发送数据的主机以一个 FIN 命令来结束数据连接，这个 FIN 命令需要另一台主机以 ACK 确认，另一台主机也发送一个 FIN 命令，这个 FIN 命令同样需要发送数据的主机以 ACK 确认；

(11) 客户端能在控制连接上发送更多的命令，这可以打开和关闭另外的数据连接；有时候客户端结束后，客户端以 FIN 命令来关闭一个控制连接，服务器端以 ACK 包来确认客户端的 FIN，服务器同样也发送它的 FIN，客户端用 ACK 来确认。

四、系统功能描述及软件模块划分

本系统包括登录、传输操作和文件操作三个模块。各模块的划分和功能如下图所示。

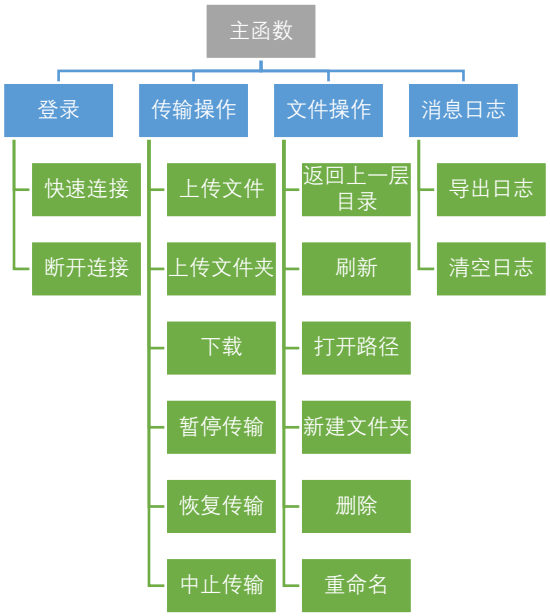

图 1 系统模块图

各模块功能简要表述：

1. **主函数**：应用程序的主入口点，负责调用初始化的主界面，通过主界面触发系统的各个功能。
2. **快速连接**：连接到远程 FTP 服务器，用户需要输入主机的 IP 地址、端口号、用户名和密码，并选择协议类型和传输模式，连接成功后系统将会打开 FTP 服务器配置中指定的目录，得到该目录下的所有文件及文件夹的详细信息和当前目录路径，并将连接过程打印到消息日志中。
3. **断开连接**：断开当前已建立的连接，清除列表视图和消息日志上的所有信息。
4. **上传文件**：打开一个选择文件对话框，对话框提示选择要上传的文件，点击“打开”按钮后，系统根据文件后缀名自动选择传输方式，将文件上传到 FTP 服务器上，同时进度条显示传输进度。传输完成后，系统自动刷新当前目录的文件列表视图并更新消息日志。
5. **上传文件夹**：打开一个选择文件夹对话框，对话框提示选择要上传的文件夹，点击“选择文件夹”按钮后，系统将选择的文件夹上传到 FTP 服务器上，同时进度条显示传输进度。传输完成后，系统自动刷新当前目录的文件列表视图并更新消息日志。
6. **下载**：用户应先选择文件列表视图中要下载的多个或单个文件和文件夹，然后单击下载按钮，系统将会打开一个文件夹浏览器对话框，对话框提

示用户选择下载目录，系统根据文件后缀名自动选择传输方式，将文件和文件夹下载到被选择的本地目录上。传输完成后，系统自动刷新当前目录的文件列表视图并更新消息日志。

7. **暂停传输：**挂起正在进行中的上传或下载线程。
8. **恢复传输：**恢复被挂起的上传或下载线程。
9. **中止传输：**关闭与服务器的数据连接，并结束上传或下载文件的线程。
10. **返回上一层目录：**打开当前目录的上一层目录，刷新文件列表视图并更新消息日志。
11. **刷新：**刷新当前目录的文件列表视图并更新消息日志。
12. **打开路径：**用户通过更改路径文本框中的内容，然后单击打开路径按钮，系统将会打开路径文本框中的目录，刷新该目录的文件列表视图并更新消息日志。
13. **新建文件夹：**打开一个新建文件夹界面，用户输入新文件夹的名称，确定后系统会在当前目录下创建一个新文件夹，刷新文件列表视图并更新消息日志。
14. **删除：**用户应先选择要删除的多个或单个文件和文件夹，然后单击删除按钮，系统将会删除用户指定的文件和文件夹，刷新文件列表视图并更新消息日志。
15. **重命名：**用户应先选择要重命名的单个文件或文件夹，然后单击重命名按钮，系统将会打开一个重命名界面，用户输入新的名称，确定后系统会更改用户指定的文件或文件夹的名称，刷新文件列表视图并更新消息日志。
16. **导出日志：**打开一个保存文件对话框，用户选择将要保存的目录后，点击“保存”按钮，当前消息日志中的文本将会保存至指定目录下。
17. **清空日志：**清空当前消息日志中的文本。

五、设计步骤

(一)软件体系结构设计

本系统采用分层设计的思想，使用三层架构，从下至上分别为：服务层、业务逻辑层、表示层。

1. **服务层：**直接与远程 FTP 服务器进行通信，负责发送命令、接收响应消息、上传和下载数据等工作。
2. **业务逻辑层：**该层是对服务层的封装，通过调用服务层中的方法来对数据业务逻辑处理，完成各种功能。
3. **表示层：**展现给用户的交互式操作界面，通过捕获用户的输入信息来触发对应的事件，事件中调用业务逻辑层的方法来完成相应的功能。

(二)主要功能模块的核心实现代码及流程图

1. 快速连接

代码文件：ftpeventlayer.cs

代码：

```
1.  public bool QuickConnect(ref string[] filelist, ref string workingdir, ref string syst, string hostaddress
    , string username, string password, int port, bool connect_mode)
2.  {
3.      ftpsvc.loadPreferences(hostaddress, username, password, port, connect_mode);
4.      if (ftpsvc.connectMainsocket())
5.      {
6.          ftpsvc.setUTF();
7.          if (ftpsvc.Login())
8.          {
9.              syst = ftpsvc.Syst();
10.             if(!Refresh(ref filelist, ref workingdir))
11.             {
12.                 return false;
13.             }
14.             return true;
15.         }
16.         else
17.         {
18.             return false;
19.         }
20.     }
21.     else
22.     {
23.         return false;
24.     }
25. }
```

流程图：

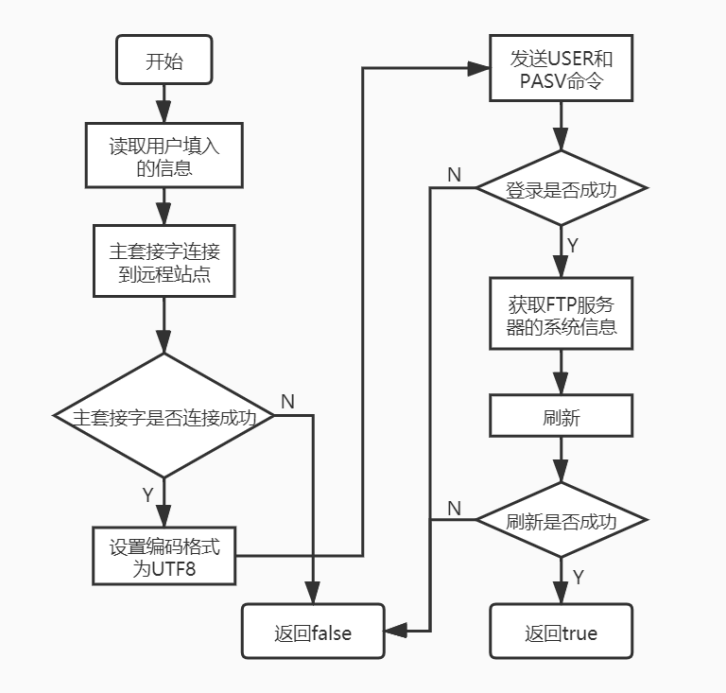

图 2 快速连接流程图

2. 上传文件

代码文件：ftpeventlayer.cs

类方法： **public bool** Upload(**ref string**[] filelist, **ref string** workingdir, **string** filename)

代码文件：ftpservice.cs

类方法：

public bool Upload(**string** filename)

代码略，详见附件中的源程序。

流程图：

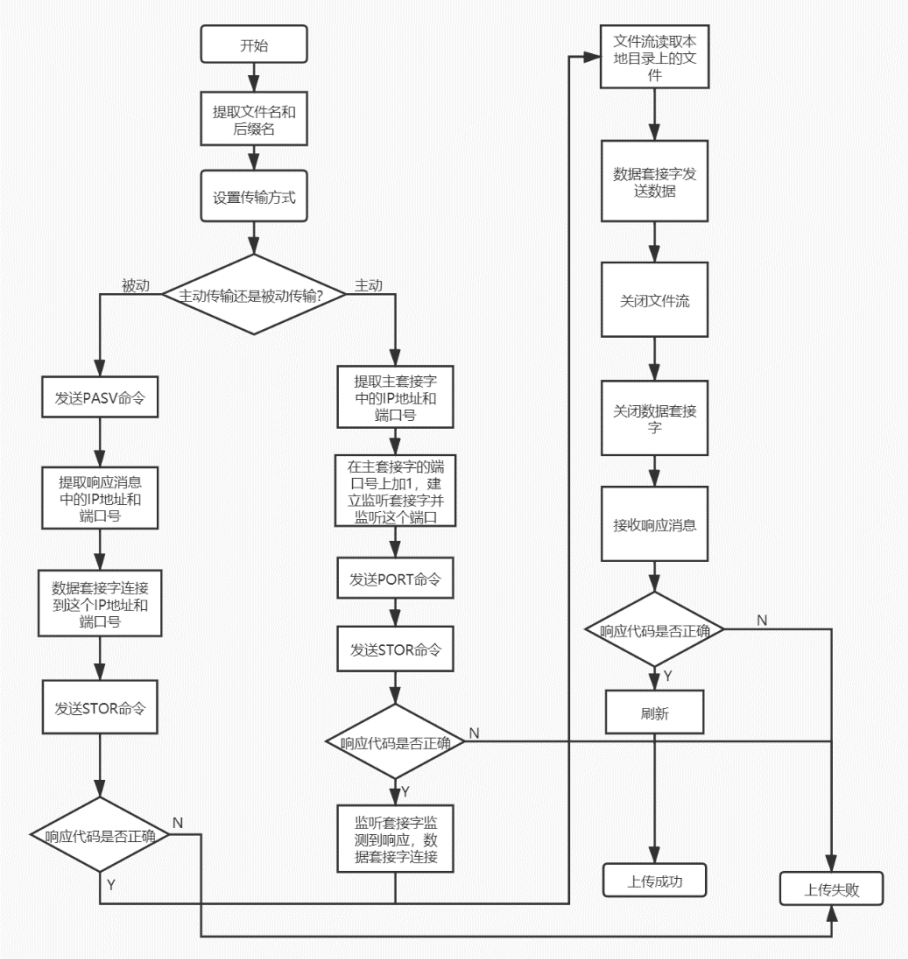

图 3 上传文件流程图

3. 上传文件夹

代码文件：ftpeventlayer.cs

代码：

```
1.  private bool uploadFolder(string localFolderPath, string remoteFolderPath)
2.  {
3.      string localFolderName = Path.GetFileName(localFolderPath);
4.      string[] allLocalFilePaths = Directory.GetFiles(localFolderPath);
5.      string[] allLocalDirectoryPaths = Directory.GetDirectories(localFolderPath);
6.
7.      ftpsvc.Mkd(localFolderName); // 新建文件夹
8.      string newRemoteFolderPath = remoteFolderPath + "/" + localFolderName;
9.      string response = ftpsvc.Cwd(newRemoteFolderPath); // 打开远程文件夹
10.     int code = ftpsvc.getResponseCode(response);
11.     if (code != 250)
12.     {
13.         return false; // 打不开文件夹则停止
14.     }
15.     if (allLocalFilePaths != null) // 上传所有文件
16.     {
17.         foreach (var filePath in allLocalFilePaths)
18.         {
```

```
            if (!Upload(filePath))
            {
                return false;
            }
        }
    }
    if (allLocalDirectoryPaths != null) // 递归
    {
        foreach (var directoryPath in allLocalDirectoryPaths)
        {
            if (!uploadFolder(directoryPath, newRemoteFolderPath))
            {
                return false;
            }
            ftpsvc.Cdup();
        }
    }
    return true;
}
public bool UploadFolder(string localFolderPath, string remoteFolderPath)
{
    if (uploadFolder(localFolderPath, remoteFolderPath))
    {
        ftpsvc.Cdup();
        return true;
    }
    else
    {
        return false;
    }
}
```

流程图：

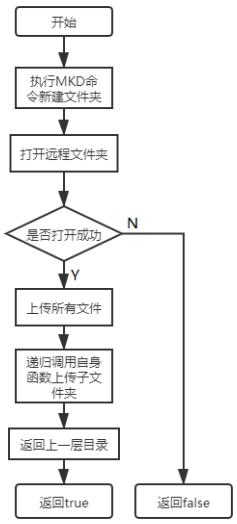

图 4 上传文件夹

4. 下载文件

代码文件：ftpeventlayer.cs

类方法：

```
public bool Download(string remote_filename, string local_filepath)

public bool DownloadFolder(string remote_folderpath, string local_filepath, string remote_filename)
```

代码略，详见附件中的源程序。

流程图：

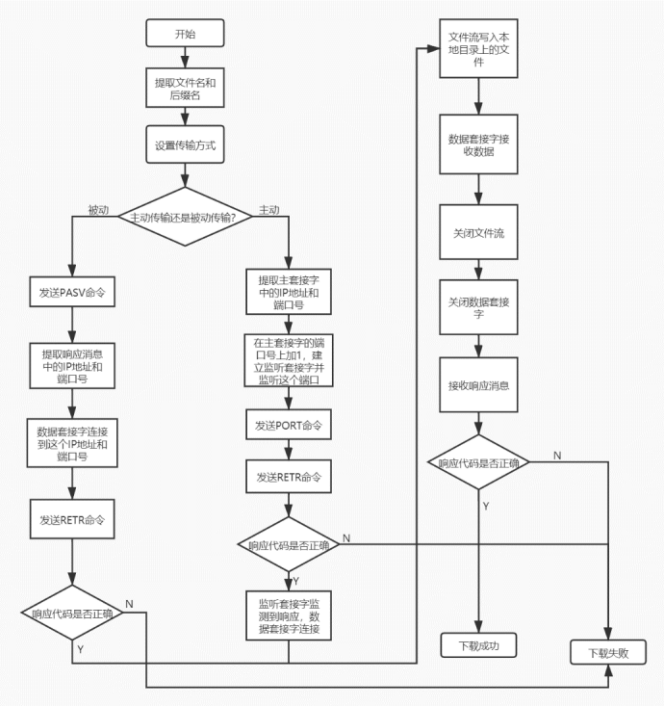

图 5 下载文件流程图

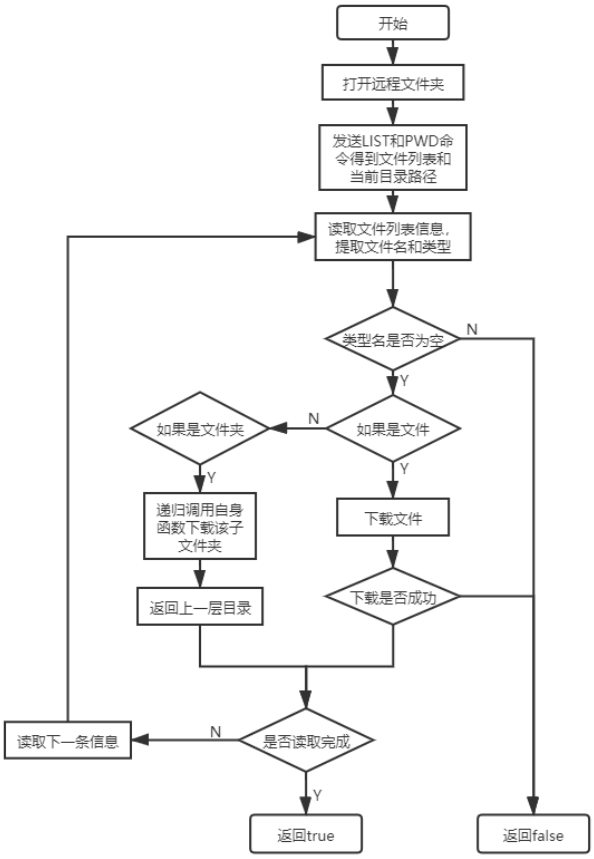

图 6 下载文件夹流程图

5. 刷新

代码文件：ftpservice.cs

代码：

```
1.  public bool Refresh(ref string[] filelist, ref string workingdir)
2.  {
3.      string rawstring = "";
4.      workingdir = ftpsvc.getDir();
5.      if (isPassive)
6.      {
7.          ftpsvc.connectDataSocketForPassive();
8.          string response1 = ftpsvc.List();
9.          int code1 = ftpsvc.getResponseCode(response1);
10.         if(code1 != 150)
11.         {
12.             return false;
13.         }
14.     }
15.     else
16.     {
17.         ftpsvc.Port();
18.         string response2 = ftpsvc.List();
19.         int code2 = ftpsvc.getResponseCode(response2);
20.         if (code2 != 150)
21.         {
```

```
            return false;
        }
        ftpsvc.connectDataSocketForActive();
    }
    rawstring = ftpsvc.getListData();
    ftpsvc.closeDataSocket();
    string response3 = ftpsvc.getResponseString();
    int code3 = ftpsvc.getResponseCode(response3);
    if (code3 != 226)
    {
        return false;
    }
    filelist = rawstring.Split("\r\n".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
    return true;
}
```

流程图：

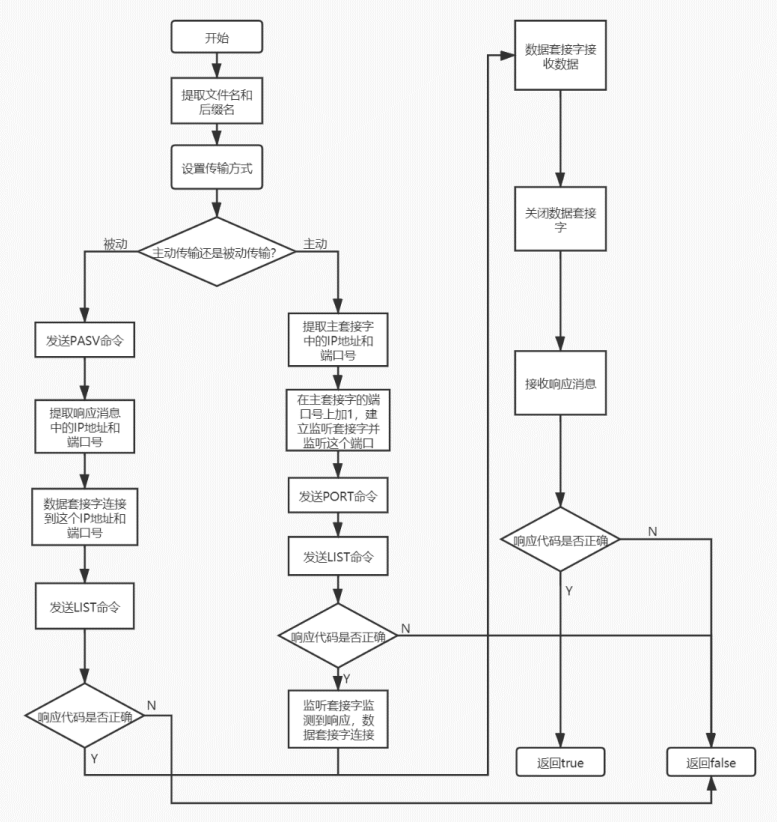

图 7 刷新流程图

6. 其他文件操作

流程图：

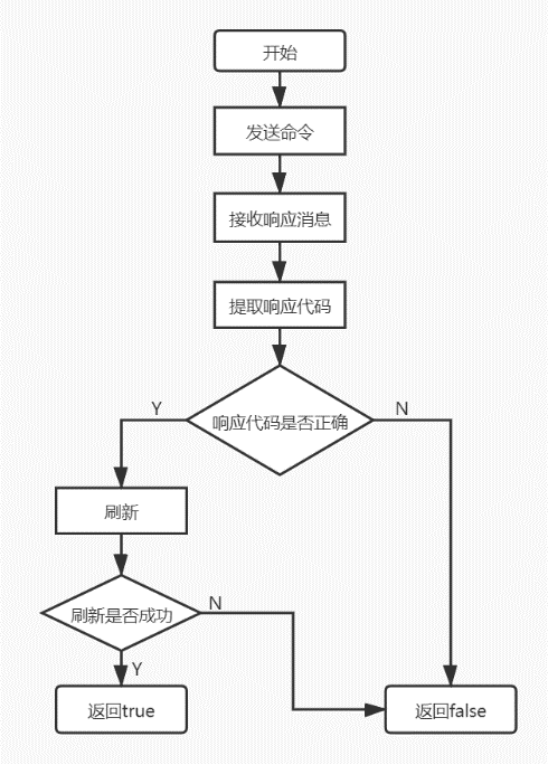

图 8 其他文件操作流程图

六、关键问题及其解决方法

1. 接收服务器返回的响应消息

问题描述：接收响应消息是一个重要的方法，每次发送命令之后都需要接收服务器传来的响应消息以进行下一步的操作。一开始编写代码时，我的做法是只要服务器传来响应报文就立即返回，没有注意到有问候消息。我在自己搭建的服务器上没有出现这个问题，因为服务器每次只返回一条响应报文，后来在连接一个公开服务器时遇到了这个问题，当发出下一条命令的时候，得到的是上一条命令的问候消息报文，导致程序运行出错。

解决方法：问候消息的特点是其响应代码的下一个字符是减号，如图 9 所示，而有效的消息的响应代码的下一个字符是空格，有效的消息会在问候消息最后出现。需要注意的是，服务器在发送响应报文时并不是一次性全部发送完毕，有时它发来的报文中没有有效的消息（即这次发来的报文全部都是问候消息），有效的消息只在最后一条报文的最后一行中。根据这些信息，解决这个问题就不是很困难了，可以检测每次服务器发来的消息中第四个字符是不是空格，如果是则说明这条消息是有效的；如果不是就先检查它发了几行消息，如果只有一行那就肯定不存在有效的消息，如果有多行，那就判断最后

一行的第四个字符是不是空格，是空格就直接返回，不是空格则继续下一次循环。

图 9 问候消息示例

仍存在的问题：这个方法仍然不够完善，比如当网络不稳定时，服务器发来的响应报文丢失，则程序会一直收不到数据进入死循环，解决方法是引入超时机制或者限制循环次数，这样做的后果是如果超时时间过短会因为收不到响应报文而导致程序错误，如果超时时间过长会使运行变慢或者接收到了其他命令的响应报文。最后考虑到服务端一般会有超时重发机制，出错的概率很小，就不再做修改。

代码：

```
1.  public string getResponseString()
2.  {
3.      string response = "";
4.      for (; ; )
5.      {
6.          Byte[] bytes = new byte[BUFFER_SIZE];
7.          int rlen = 0;
8.          rlen = mainsocket.Receive(bytes, bytes.Length, 0);
9.          string temp = Encoding.UTF8.GetString(bytes).Replace("\0", string.Empty);
10.         logmessage += temp;
11.         response = temp;
12.         if (temp.Substring(3, 1) != " ") //如果响应消息的第四个位置不是空格，则说明这是一条问候消息
13.         {
14.             string[] lines = temp.Split('\n');
15.             if(lines.Length > 2)
16.             {
17.                 if(lines[lines.Length - 2].Substring(3, 1).Equals(" ")) //判断倒数第二个字符串的响应代码后是否包含空格
18.                 {
19.                     response = lines[lines.Length - 2] + "\n";
```

```
                    break;
                }
            }
            continue;
        }
        if (rlen < bytes.Length)
        {
            break;
        }
    }
    return response;
}
```

流程图：

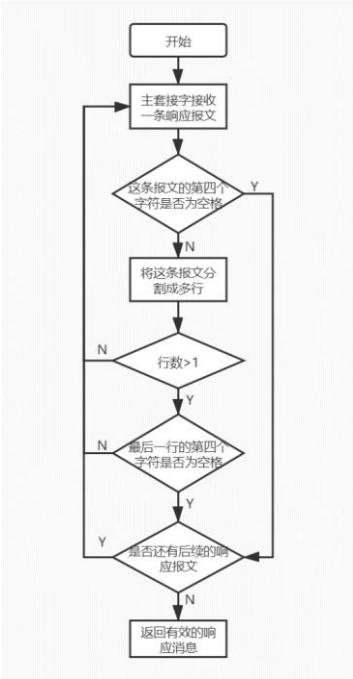

图 10 接收响应消息流程图

2. 数据套接字接收列表数据的缓存策略

问题描述：这个方法是用于接收服务器对 LIST 命令响应的列表数据。在程序的实际运行过程中，由于网络延迟，当列表数据较多的时候，它会出现系统无法接收到完整的数据的情况，导致在读取数据的时候出错，无法输出正确的列表信息。

解决方法：引入逐次加倍扩大缓冲区大小的策略和进程等待机制。如果第一次接收数据的缓存已满，则在第二次将缓冲区大小扩充到两倍，并等待 1 秒钟保证数据传输完成，依此类推。这尽管会带来一定的程序运行延迟，但降低了错误率，提高了系统的鲁棒性。

代码：

```
1.  public string getListData()
2.  {
3.      string data = "";
4.      int times = 1;
5.      for (; ; )
6.      {
7.          Byte[] bytes = new Byte[times * BUFFER_SIZE];
8.          int rlen = datasocket.Receive(bytes, bytes.Length, 0);
9.          data += Encoding.UTF8.GetString(bytes);
10.         times *= 2;
11.         if (rlen < bytes.Length)
12.         {
13.             break;
14.         }
15.         Thread.Sleep(1000); //进程等待 1 秒钟，尽量保证数据传输完成
16.     }
17.     return data.Replace("\0", string.Empty);
18. }
```

流程图：

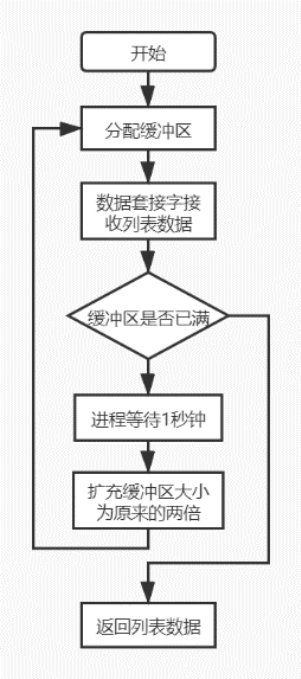

图 11 数据套接字接收列表数据流程图

七、软件使用说明

用户在使用本软件时，先单击工具栏的快速连接按钮或在菜单栏中选择文件→快速连接，系统会打开快速连接界面，用户在文本框中输入目标服务器的 IP 地址和端口号（目前支持连接的目标服务器操作系统内核有 Unix 和 Windows_NT），并输入用户名与密码或者点击匿名登录复选框，再选择传输模式（如果使用主动模式，需要关闭防火墙，推荐使用被动模式），然后单击快速连接按钮登录到远

程 FTP 服务器。

登录成功后，用户可以单击工具栏中的按钮或菜单栏的选项完成各种操作，还可以双击列表视图中的文件夹快速打开选中目录或上一层目录。

当用户想要退出时，可以单击菜单栏中的文件→退出，或者单击标题栏的退出按钮，关闭软件。

八、参考资料

[1] FTP 协议详解. https://www.jianshu.com/p/dd11c8e3c27c

[2] FTP Port 模式和 FTP Passive 模式 详细介绍. https://blog.csdn.net/astro_boy/article/details/84089907?depth_1-utm_source=distribute.pc_relevant.none-task&utm_source=distribute.pc_relevant.none-task

[3] RFC 959 (Standard) File Transfer Protocol (FTP). Postel, J. & Reynolds, J. (October 1985).