CLASS EPTIPI

MENU

– Các hàm hỗ trợ LIST và NLST

CÁC HẰNG SỐ VÀ SUBCLASS

- **enum** `FTPCode`
  - Các hằng số mã trả về cơ bản của **ftp**

```
enum FTPCode {
    CONNECT_SUCCESS = 220,

    LOGIN_SUCCESS   = 230,
    LOGIN_FAILED    = 530,

    OPEN_PASV_PORT  = 227,
    OPEN_LPSV_PORT  = 228,
    OPEN_ESPV_PORT  = 229,

    COMMAND_SUCCESS = 200,
    CANNOT_OPEN_DATA_CONNECT = 425,

    DATA_ALREADY_OPEN   = 125,
    READY_TRANSFER      = 150,
    TRANSFER_SUCCESS    = 226,

    CWD_SUCCESS = 250,

    FILE_STATUS = 213,

    DISCONNECT = 421
};
```

- **namspace** `FTPDataMode`
  - Dùng để khởi tạo Hằng số các trạng thái ACTIVE và PASSIVE

```
namespace FTPDataMode {
    const UCHAR PASSIVE = 0;
    const UCHAR ACTIVE = 1;
    const UCHAR DEFAULT = PASSIVE;
}
```

- **namspace** `FTPFileMode`
  - Dùng để khởi tạo Hằng số các trạng thái ACTIVE và PASSIVE

```
namespace FTPFileMode {
    const UCHAR BINARY = 0;
    const UCHAR ASCII = 1;
    const UCHAR DEFAULT = BINARY;
}
```

- **BUFFER_LENGTH** = 512
  - Độ lớn mặc định của buffer dùng để đọc thông tin server gửi về

- **struct** CallbackInfo
  - Dùng để truyền dữ liệu cần thiết khi gọi Eptipi::openDataPort

```
struct CallbackInfo {
    std::string path = "";
    Eptipi * mainFTP = NULL;
    CSocket * dataCon = NULL;
    UINT64 filesize = 0;
};
```

CLASS MEMBER

- **CSocket** cmdConn
  - Sử dụng để kết nối đến control port của server (port 21)
  - Gửi lệnh và nhận phản hồi
- **const wchar_t** servername
  - Lưu lại tên server dùng sử dụng lại.
- **int** returnCode
  - Lưu lại mã phản hồi từ server (VD: 220, 530, …)
- **std::string** returnStr
  - Lưu lại chuỗi phản hồi từ server (VD: 200 OK)
- **int** returnPort
  - Lưu lại thông tin port trả về từ server (nếu có)
  - -1 nếu không có thông tin port trả về.

MAIN FUNCTION

- `Eptipi::Eptipi()`
  Khởi tạo class Eptipi.

  Các member trong class sẽ có giá trị mặc định.

  Chưa kết nối với server nào hết.

- `void Eptipi::connectServer(const wchar_t* serverAddr)`
  Kết nối đến server có địa chỉ `serverAddr`, port 21

  Throw **exception** khi không thể kết nối.

- `bool Eptipi::login()`
  Tạo prompt đăng nhập (username/password) trên màn hình **console**

  Trả về **false** khi đăng nhập thất bại

- `void Eptipi::handleCmd(std::string cmd, std::string path)`
  Thực thi theo lệnh truyền vào `cmd`, với các tham số lệnh trong chuỗi `path`

  Để biết các lệnh `cmd` hợp lệ, xem Eptipi::showAllCmd()

- `void Eptipi::showAllCmd()`
  Liệt kê tất cả các `command` hợp lệ của chương trình lên màn hình console

---

CÁC HÀM BÊN DƯỚI

- `void Eptipi::lietKeChiTiet()`
  In ra màn hình danh sách **chi tiết** directory của path hiện tại trên server
  - Sử dụng phương thức active hoặc passive
  - Sử dụng lệnh LIST của **ftp protocal command**
  - In ra màn hình console

  Sử dụng:

```
Eptipi FTP;
///....
///da connect va login
FTP.lietKeChiTiet("*.txt"); //liet tat ca cac file txt
///...
```

- `void Eptipi::lietKeDonGian()`
  In ra màn hình danh sách tên của thư mục và file ở directory hiện tại trên server
  - Sử dụng passive hoặc active
  - Sử dụng lệnh NLST của **ftp protocal command**
  - In ra màn hình console

  Sử dụng:

```
Eptipi FTP;
///....
///da connect va login
FTP.lietKeDonGian("*.txt"); //liet tat ca cac file txt
///...
```

- `void Eptipi::lietKeClientChiTiet()`
  In ra màn hình danh sách tên của thư mục và file ở directory hiện tại trên client
  - Sử dụng lệnh system của STL (trên windows)
  - In ra màn hình console

- `void Eptipi::lietKeClientDonGian()`
  In ra màn hình danh sách tên của thư mục và file ở directory hiện tại trên client
  - Sử dụng lệnh system của STL (trên windows)

- In ra màn hình console

- **`void Eptipi::changeServerDir(std::string path)`**
  Thay đổi đường dẫn trên server

- **`void Eptipi::changeClientDir(std::string path)`**
  Thay đổi đường dẫn ở client

- **`void Eptipi::downFile(std::string path)`**
  Download file có đường dẫn `path` trên server về folder hiện hành của client

  ```
  this->downFile("a.txt");
  ```

- **`void Eptipi::upFile(std::string path)`**
  Upload một file có đường dẫn `path` ở client lên server

  ```
  this->upFile("a.rar");
  ```

- **`void Eptipi::downNhieuFile(std::string path)`**
  Download những file có đường dẫn trong `path` từ server về client

  ```
  this->downNhieuFile("*.txt a.rar b.zip *.mp*");
  // *.txt    - tat ca file text
  // a.rar b.zip  - 2 file co ten cu the
  // *.mp*    - tat ca file nhac, video (mp3, mp4)
  ```

- **`void Eptipi::upNhieuFile(std::string path)`**
  Upload những file có đường dẫn trong `path` từ client lên server

  ```
  this->upNhieuFile("*.txt a.rar b.zip *.mp*");
  // *.txt        - tat ca file text
  // a.rar b.zip      - 2 file co ten cu the
  // *.mp*        - tat ca file nhac, video (mp3, mp4)
  ```

CÁC HÀM HỖ TRỢ BÊN DƯỚI NỮA

- `public void sendCmd(*std::string* cmd)`
  Gửi chuỗi lệnh raw lên server thông qua **cmdConn**

```
cmdConn.Send(cmd.c_str(), cmd.length());
```

- `public void sendCmd(*std::wstring* cmd)`
  Tương tự như trên

- `public void receiveAll()`
  Nhận các phản hồi trên server xuống rồi sau đó truyền vào **returnStr**

- `public void receiveOneLine()`
  Nhận 1 phản hồi (1 dòng có kí tự cuối là *) từ server xuống và xử lý phản hồi đó thành **returnCode**, **returnStr** và **returnPort**(nếu có). **VD**:
  - **buffer** = 200 Command OK
    - `returnCode = 220`
    - `returnStr = "200 Command OK"`
    - `returnPort = -1`
  - **buffer** = 227 Entering Passive Port (127,0,0,1,25,5)
    - `returnCode = 227`
    - `returnStr = "227 Entering Passive Port (127,0,0,1,25,5)"`
    - `returnPort = 25*256 + 5 = 6405`

- `public int getCode()`
  Trả về **returnCode**

- `public std::string getReturnStr()`
  Trả về **returnStr**

- `public int getReturnPort()`
  Trả về **returnPort**

- `std::vector<std::string> getLIST(std::string path)`
  Sử dụng như lệnh dir, lưu kết quả vào vector, không in ra

- `std::vector<std::string> getNLST(std::string path)`
  Sử dụng như lệnh ls, lưu kết quả vào vector, không in ra

CÁC HÀM HỖ TRỢ MỞ DATA CONNECTION

- `protected CSocket * openPassivePortAndConnect();`
  Mở một Passive connection và trả về địa chỉ của Socket vừa kết nối.

  Hàm bao gồm thực hiện việc yêu cầu kết nối passive đến server (PASV và ESPV và LSPV)

  Sau đó tạo CSocket kết nối thông qua port lấy từ phản hồi của server, trả về địa chỉ

- `protected CSocket * openActivePortAndConnect();`
  Mở một active connection và trả về địa chỉ của port mà client tự mở

  Hàm bao gồm tạo CSocket và gửi thông tin port đó lên server thông qua lệng PORT hoặc EPRT

  Trả về địa chỉ CSocket của port mà client mở (chưa tìm đc sock của server)

- `protected void openDataPort(bool (*before)(*CallbackInfo*&), void (*after)(*CallbackInfo*&), *CallbackInfo*&);`
  Để thực hiện thao tác trên passive connection, ta thao tác trực tiếp lên socket vừa tạo ra từ *openPassivePortAndConnect()*

  ```
  1. Tại socket
  2. Gửi lệnh (LIST, RETR, ...)
  3. Thao tác trên socket đó
  ```

  Tuy nhiên, khi trên active connection, socket trả ra thừ *openActivePortAndConnect()* chỉ là socket để nghe, chưa phải để truyền dữ liệu nên ta phải chờ 1 thao tác lệnh nữa mới có thể Listen và Accept connect từ server. Tức là:

  ```
  1. Tạo port
  2. Gửi lệnh (LIST hay NLST gì đó)
  3. Accept connect từ server -> socket để truyền mới
  4. Thao tác trên socket vừa accept ở bước 3
  ```

  Do đó, việc chen ngang ở bước 3 đã khiến ta phải làm thủ công 1 bước nữa để có thể tìm đc socket đc accept. Cho nên, với nhiều hàm cần sử dụng lại, ta lại viết lại khá mệt.

  Để giải quyết, ta viết 1 hàm để nhận vào hành động trước và sau khi accept, việc accept đã đc giấu vô hàm, ko cần viết lại
  - **@param** `bool (* before)(CallbackInfo&)` nhận vào hàm thực hiệu trước khi accept server, VD: Gửi LIST, NLST,... Trả về false khi việc connect không thành công
  - **@param** `void (* after)(CallbackInfo&)` nhận vào hàm thực hiện sau khi accept đc server, lúc này socket được accept sẽ được truyền vào CallbackInfo

- **@param** `CallbackInfo*` là biến lưu lại giá trị cần thiết trong quá trình gọi lại hàm before và after

- ## CÁCH SỬ DỤNG

1. Viết 1 struct để khởi tạo 2 hàm before và after, nhận vào tham biến CallbackInfo
2. Sau đó khởi tạo CallbackInfo nào đó, chứa thông tin cần thiết
3. Truyền 3 tham số trên và gọi hàm openDataPort

```
// Vi du cho lenh `dir`
struct deKhaiBaoHam {
    static bool before(CallbackInfo &cb) {
        cb.mainFTP->sendCmd("LIST "+cb.path+"\r\n");
        cb.mainFTP->receiveStatus();
        cout << '\t' << cb.mainFTP->getReturnStr() << endl;

        if (cb.mainFTP->getCode() != FTPCode::READY_TRANSFER
            && cb.mainFTP->getCode() != FTPCode::DATA_ALREADY_OPEN)
            return false;
        return true;
    };

    static void after(CallbackInfo &cb) {
        if (cb.dataCon == NULL) return;

        char buffer[BUFFER_LENGTH];
        memset(buffer, 0, BUFFER_LENGTH);
        while (cb.dataCon->Receive(buffer, BUFFER_LENGTH - 1) > 0)
{
            cout << buffer;
            memset(buffer, 0, BUFFER_LENGTH);
        }
    }
};

this->sendCmd("TYPE A\r\n"); //ascii mode
this->receiveStatus();

CallbackInfo cb;
cb.path = path;
cb.mainFTP = this;
openDataPort(deKhaiBaoHam::before, deKhaiBaoHam::after, cb);
```