#

CLASS EPTIPI

* [CLASS MEMBER](#class-member)
* [BASIC FUNCTION USED FOR ALL](#basic-function)
* [MAIN FUNCTION](#main-function)
  + [Construct](#void-eptipi--eptipi)
  + [Connect to server](#void-eptipi--connectserverconst-wchar-t-serveraddr)
  + [Login](#bool-eptipi--login)
* [OPEN DATA PORT](#open-data-port)

## CÁC HẰNG SỐ VÀ SUBCLASS

* **enum** FTPCode
  + Các hằng số mã trả về cơ bản của **ftp**
  + 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,  
    READY\_TRANSFER = 150,  
    TRANSFER\_SUCCESS = 226,  
    FILE\_STATUS = 213,  
    CONNECT\_FAILED = 421
* **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](#open-data-port)
  + struct CallbackInfo {  
     std::string path = "";  
     Eptipi \* mainFTP = NULL;  
     CSocket \* dataCon = NULL;  
     int 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.

## BASIC FUNCTION

### 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**

## 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)

### 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

### 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

### void Eptipi::lietKeClientChiTiet()

### void Eptipi::lietKeClientDonGian()

### void Eptipi::changeServerDir(std::string path)

### void Eptipi::changeClientDir(std::string path)

### void Eptipi::showAllCmd()

* Liệt kê tất cả các command hợp lệ của chương trình lên màn hình console

## OPEN DATA PORT

## 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\r\n");  
     cb.mainFTP->receiveOneLine();  
     cout << '\t' << cb.mainFTP->getReturnStr() << endl;  
     if (cb.mainFTP->getCode() != FTPCode::READY\_TRANSFER)  
     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);  
     }  
     }  
    };  
      
    //ascii mode  
    this->sendCmd("TYPE A\r\n");   
    this->receiveOneLine();  
      
    // thong tin truyen vao callback  
    // truyen this vao mainFTP de su dung lai cac ham viet san  
    CallbackInfo cb;  
    cb.mainFTP = this;  
      
    //goi ham  
    openDataPort(deKhaiBaoHam::before, deKhaiBaoHam::after, cb);