**1. 引言**

**1.1 目的**

本文档旨在提供libftp模块的概要设计，包括系统结构、模块组成、功能实现等，以便开发和维护该模块。

**1.2 范围**

libftp是一个轻量级的FTP库，用于实现FTP客户端的功能，包括连接FTP服务器、文件传输、目录操作等。

**1.3 术语和缩略语**

* **FTP**：File Transfer Protocol
* **SSL**：Secure Sockets Layer

**1.4 参考资料**

* [libftp项目GitHub地址](https://github.com/deniskovalchuk/libftp)

**2. 总体描述**

**2.1 产品透视图**

libftp模块是一个独立的库，可以集成到各种C++项目中，实现FTP客户端的功能。

**2.2 功能描述**

libftp模块的主要功能包括：

* 连接FTP服务器
* 用户身份验证
* 文件上传与下载
* 文件与目录操作
* 支持SSL/TLS安全连接

**2.3 用户特点**

libftp模块面向需要FTP客户端功能的开发者和应用程序。

**2.4 约束**

* 支持的编译器：GCC、Clang、MSVC
* 依赖库：Boost.Asio、OpenSSL

**3. 设计约束**

**3.1 假设和依赖**

* 依赖Boost.Asio库进行网络通信
* 依赖OpenSSL库进行SSL/TLS加密

**3.2 系统环境**

* 支持的操作系统：Linux、Windows、macOS

**4. 系统架构设计**

**4.1 模块划分**

libftp模块主要包括以下几个子模块：

* 连接模块（FTPConnection）
* 命令模块（FTPCommand）
* 响应模块（FTPResponse）
* 异常处理模块（FTPException）
* 文件传输模块（FTPTransfer）
* 安全连接模块（FTPSSL）

**4.2 模块关系图**

plantuml

复制代码

@startuml

interface StreamAdapter

interface SocketInterface

class SocketBase {

+ initialize()

+ cleanup()

}

class Socket extends SocketBase {

- boost::asio::ip::tcp::socket socket

+ open()

+ close()

}

class SSLSocket extends Socket {

- SSL\* ssl

+ connect()

+ disconnect()

}

class ControlConnection extends Socket {

- boost::asio::ip::tcp::socket socket

- std::shared\_ptr<boost::asio::io\_context> io\_context

+ open(const std::string& server, int port)

+ close()

+ sendCommand(const std::string& command)

+ readResponse(): std::string

}

class DataConnection extends Socket {

- boost::asio::ip::tcp::socket socket

- std::shared\_ptr<boost::asio::io\_context> io\_context

+ open(const std::string& server, int port)

+ close()

+ sendData(const std::string& data)

+ receiveData(): std::string

}

class FTPStreamInput implements StreamAdapter {

- std::istream& stream

+ FTPStreamInput(std::istream& stream)

+ read(char\* buffer, std::size\_t size): std::size\_t

}

class FTPStreamOutput implements StreamAdapter {

- std::ostream& stream

+ FTPStreamOutput(std::ostream& stream)

+ write(const char\* buffer, std::size\_t size): std::size\_t

}

SocketInterface <|-- Socket

StreamAdapter <|-- FTPStreamInput

StreamAdapter <|-- FTPStreamOutput

class FTPClient {

- std::string server

- int port

- std::string user

- std::string password

+ connect(const std::string& server, int port, const std::string& user)

+ login(const std::string& user, const std::string& password)

+ listFiles(const std::string& directory): std::vector<std::string>

+ uploadFile(const std::string& localFilePath, const std::string& remoteFilePath)

+ downloadFile(const std::string& remoteFilePath, const std::string& localFilePath)

+ renameFile(const std::string& oldName, const std::string& newName)

+ deleteFile(const std::string& filePath)

+ disconnect()

}

FTPClient --> FTPConnection

FTPClient --> FTPCommand

FTPClient --> FTPResponse

FTPClient --> FTPException

FTPClient --> FTPReply

FTPClient --> FTPTransferCallback

FTPClient --> FTPStreamInput

FTPClient --> FTPStreamOutput

FTPClient --> FTPObserver

FTPClient --> FTPSSL

FTPClient --> FTPTransferMode

FTPClient --> FTPTransferType

class FTPConnection {

- boost::asio::ip::tcp::socket socket

- std::shared\_ptr<boost::asio::io\_context> io\_context

+ open(const std::string& server, int port)

+ close()

+ sendCommand(const std::string& command)

+ readResponse(): std::string

}

FTPConnection --> ControlConnection

FTPConnection --> DataConnection

class FTPCommand {

+ std::string command

+ FTPCommand(const std::string& cmd)

+ getCommand(): std::string

}

class FTPResponse {

- int code

- std::string message

+ FTPResponse(int code, const std::string& message)

+ getCode(): int

+ getMessage(): std::string

}

class FTPException {

+ FTPException(const std::string& message)

}

class FTPReply {

- int code

- std::string text

+ FTPReply(int code, const std::string& text)

+ getCode(): int

+ getText(): std::string

}

class FTPTransferCallback {

+ onTransferStart()

+ onTransferProgress(size\_t transferred)

+ onTransferComplete()

}

class FTPObserver {

+ onNotify()

}

class FTPSSL {

- SSL\_CTX\* ctx

+ initialize()

+ cleanup()

}

class FTPTransferMode {

+ std::string mode

}

class FTPTransferType {

+ std::string type

}

class Utils {

+ formatCommand(const std::string& cmd): std::string

+ parseResponse(const std::string& response): std::pair<int, std::string>

}

FTPCommand --> Utils

FTPResponse --> Utils

class IStreamAdapter {

- std::istream& stream

+ IStreamAdapter(std::istream& stream)

+ read(char\* buffer, std::size\_t size): std::size\_t

}

class OStreamAdapter {

- std::ostream& stream

+ OStreamAdapter(std::ostream& stream)

+ write(const char\* buffer, std::size\_t size): std::size\_t

}

FTPStreamInput --> IStreamAdapter

FTPStreamOutput --> OStreamAdapter

class NetContext {

- boost::asio::io\_context io\_context

}

class NetUtils {

+ resolve(const std::string& host): std::string

}

ControlConnection --> NetContext

DataConnection --> NetContext

NetUtils --> NetContext

@enduml

**4.3 模块功能描述**

* **FTPClient**：管理FTP服务器连接和文件操作。
* **FTPConnection**：封装网络连接。
* **FTPCommand**：表示FTP命令。
* **FTPResponse**：表示FTP响应。
* **FTPException**：处理FTP相关异常。
* **FTPReply**：表示FTP回复。
* **FTPTransferCallback**：处理文件传输回调。
* **FTPStreamInput**和**FTPStreamOutput**：处理文件输入和输出流。
* **FTPSSL**：管理SSL连接。
* **Utils**：提供命令格式化和响应解析工具。

**4.3 模块功能描述**

时序图

连接和身份验证时序：

连接和验证FTP服务器的时序图

plantuml

复制代码

@startuml

actor User

participant "FTPClient" as client

participant "FTPConnection" as connection

User -> client: connect(server, port)

client -> connection: openConnection(server, port)

connection -> client: connection established

User -> client: login(user, password)

client -> connection: sendCommand("USER " + user)

connection -> client: response("331 User name okay, need password.")

client -> connection: sendCommand("PASS " + password)

connection -> client: response("230 User logged in, proceed.")

@enduml

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. 活动图

文件上传活动图：

文件上传过程的活动图

plantuml

复制代码

@startuml

start

:用户选择要上传的文件;

:建立FTP连接;

if (连接成功?) then (yes)

:发送"STOR"命令;

:传输文件数据;

:接收成功响应;

else (no)

:显示连接错误;

endif

:关闭FTP连接;

stop

@enduml

**4.4 接口设计**

* **FTPClient接口**：

cpp

复制代码

class FTPClient {

void connect(const std::string& server, int port);

void login(const std::string& user, const std::string& password);

std::vector<std::string> listFiles(const std::string& directory);

void uploadFile(const std::string& localFilePath, const std::string& remoteFilePath);

void downloadFile(const std::string& remoteFilePath, const std::string& localFilePath);

void renameFile(const std::string& oldName, const std::string& newName);

void deleteFile(const std::string& filePath);

void disconnect();

};

**5. 数据库设计**

libftp模块不涉及数据库设计。

**6. 异常处理**

libftp模块通过**FTPException**类处理异常。所有异常均通过抛出**FTPException**对象来报告。

**7. 安全性设计**

libftp模块通过**FTPSSL**类支持SSL/TLS加密连接，确保数据传输的安全性。

**8. 性能需求**

libftp模块在设计时考虑了性能需求，采用异步网络通信提高效率。

**9. 其他设计说明**

libftp模块的设计遵循模块化、可扩展性和高性能的原则。未来可能增加更多的功能和优化。