简易FTP服务器

> **Important**
>
> - 本项目全部使用Cangjie编程语言进行开发，设计思路参考MinimalFTP，原项目采用Java进行开发，本项目采取的木兰开源协议与MinimalFTP采用的Apache-2.0相兼容。
> - 本项目于2025.2.14日起，推翻原先设计重新编写，以便更好地兼容市面上主流的FTP客户端软件。
> - 演示视频可参考项目根目录下演示视频
> - 本文档仅介绍功能，更加详细的设计实现请参考项目开发文档

项目测试环境：

- 服务端：openEuler操作系统
- 客户端：Ubuntu操作系统
- 两台机器均为云服务器且Ubuntu自带FTP客户端软件

🚨安装

项目根目录下提供了一键安装启动脚本，但请注意，为了确保环境变量生效，**❗❗❗请使用**`source run-ftp.sh`启动项目。

脚本功能包括：

- 检查 `cangjie` 工具链是否已安装
- 检查 `openEuler` 防火墙状态
- 编译 `FTP` 服务器项目
- 运行 `FTP` 服务器项目

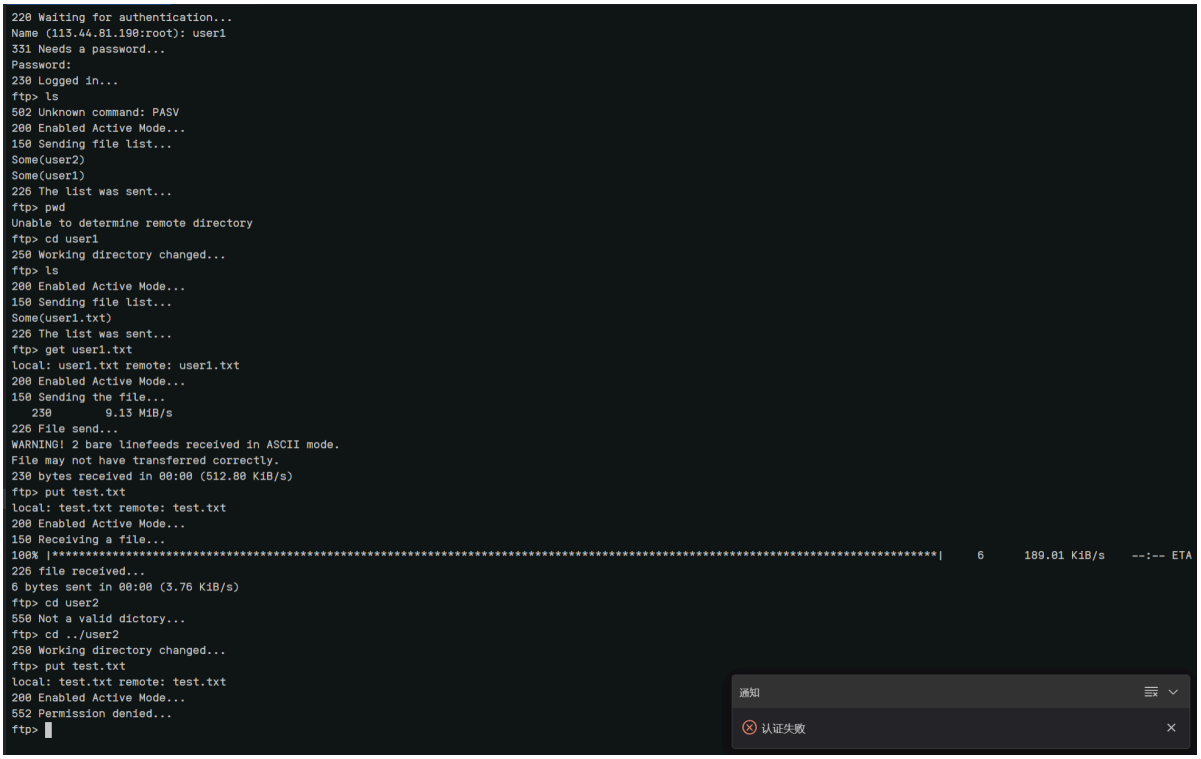
🚀功能

目前该简易FTP服务器采用主动连接的工作方式且支持以下FTP常用命令：

- USER：输入用户名
- PASS：输入用户密码
- QUIT：退出用户登录
- PORT：指定数据连接端口
- NLST：列出当前目录下的文件（Windows上FTP客户端适用）
- LIST：列出当前目录下的文件（UNIX系统中使用）
- XPWD：显示当前工作路径（Windows上客户端适用）
- STOR：上传文件
- CWD：更改服务上的工作目录
- RETR：下载文件
- 更多常用命令持续添加中.....

软件实现

❗❗❗测试下列功能时请使用两台拥有公网IP的云服务器或者将服务端与客户端起在同一台主机上。可能是由于暂未支持被动模式，本人本地电脑作为客户端，云服务器作为服务器端，测试时上传下载一直处于尝试连接状态，请注意。



端口监听功能：

关键代码：

```
// 服务端套接字
private var serverSocket: TcpServerSocket = TcpServerSocket(bindAt: 21)

public func listen() {
    // 创建服务器套接字
    // 持续接受连接以不断更新状态
    // 设置backlog大小为50
    serverSocket.backlogSize = 50
    serverSocket.bind()
    // 检查连接是否关闭
    while (!serverSocket.isClosed()) {
        update()
    }
}
```

利用仓颉为我们提供的`TcpServerSocket`，很容易就可以创建出一个采用TCP的服务端套接字。同时不断更新连接，从而实现端口的监听。

文件上传功能：主要是PORT和STOR的联合使用。

关键代码：

```
// 处理STOR指令，用以上传文件到服务器
private func stor(path: String) {
    if (!server.getAuthenticator().writePerm(username, currentDir)) {
```

```
        sendResponse(552, "Permission denied...")
        return
    }
    sendResponse(150, "Receiving a file...")
    if (File.exists(currentDir.join(path))) {
        // 文件存在，进行覆盖操作
        File.delete(currentDir.join(path))
    }
    File.create(currentDir.join(path))
    try {
        receiveData(currentDir.join(path))
        sendResponse(226, "file received...")
    } catch (e: Exception) {
        sendResponse(999, e.message)
    }
}
```

基本思路为首先判断用户是否在该目录下具有上传文件的权限，而后如果具有权限为写操作做准备，存在同名文件则进行覆盖。

文件下载功能：主要是PORT和RETR的联合使用。

关键代码：

```
// 处理RETR指令，用以从服务器下载文件
private func retr(path: String) {
    if (File.exists(currentDir.join(path))) {
        // 进行权限验证，判断是否具有读和下载的权限
        if (!server.getAuthenticator().readPerm(username, currentDir.join(path)))
{
            sendResponse(552, "Permission denied...")
            return
        }
        sendResponse(150, "Sending the file...")
        try {
            let file = File.openRead(currentDir.join(path))
            let bytes = file.readToEnd()
            sendData(bytes)
            sendResponse(226, "File send...")
        } catch (e: Exception) {
            sendResponse(999, e.message)
        }
        return
    }
    sendResponse(550, "Not a valid file...")
}
```

基本思路为首先判断是否当前目录存在该文件，存在后进行权限验证，判断该用户是否具有读取和下载的权限，如果具有权限通过主动模式进行数据的传送即可。

多会话并发功能

关键代码：

```
let thread = spawn {
    =>
```

```
    while (!client.isClosed()) {
        // 持续更新连接状态
        try {
            update()
        } catch (e: SocketException) {
            println("Connection closed by the remote host...")
            close()
        }
    }
    try {
        client.close()
    } catch (e: Exception) {
        println("An Exception happen!")
    }
}
```

核心思想为与一个客户端建立FTP连接就单独创建一个线程进行连接的处理，以此实现多会话的并发操作。同时借助于仓颉的并发功能，我们还可以通过`synchronized`等关键字实现并发控制。

匿名账户和普通用户登录功能：主要是USER和PASS的联合使用

关键代码：

```
// 处理USER指令
private func user(username: String) {
    // 已验证身份
    if (connnectHandler.isAuthenticated()) {
        sendResponse(230, "Logged in...")
        return
    }

    // 获取身份认证器实例进行验证
    let auth = server.getAuthenticator()
    if (auth.needsPassword(username)) {
        this.username = username
        sendResponse(331, "Needs a password...")
    } else {
        if (auth.authenticate(username, "")) {
            this.username = username
            connnectHandler.setAuthenticated()
            sendResponse(230, "Logged in...")
        } else {
            sendResponse(530, "Authentication failed...")
            close()
        }
    }
}

// 处理PASS指令
private func pass(password: String) {
    if (connnectHandler.isAuthenticated()) {
        sendResponse(230, "Logged in...")
    }

    // 进行身份验证
    let auth = server.getAuthenticator()
```

```
    if (auth.authenticate(username, password)) {
        connnectHandler.setAuthenticated()
        sendResponse(230, "Logged in...")
    } else {
        sendResponse(530, "Authentication failed...")
        close()
    }
}
```

基本思路首先处理USER命令，如果不是匿名用户则发送响应提示用户需要密码，再处理PASS命令进行身份的校验即可。

账户权限控制功能

核心代码：

```
// 添加文件权限
public func addPermission(username: String, filepath: Path, read: Bool, write:
Bool) {
    permList.append(Permission(username, filepath, read, write))
}

// 添加文件权限
public func addPermission(username: String, filepath: Path) {
    permList.append(Permission(username, filepath))
}

// 获取文件读权限
public func readPerm(username: String, filepath: Path) {
    for (perm in permList) {
        // 查找到相匹配的
        // 根据目录递归匹配，所以这里直接看绝对路径是否呈现出包含关系
        if (perm.username == username &&
filepath.toCanonical().toString().startsWith(perm.filepath.toCanonical().toString
())) {
            return perm.read
        }
    }
    // 查找不到一律返回false
    return false
}


// 获取文件写权限
public func writePerm(username: String, filepath: Path) {
    for (perm in permList) {
        // 同理
        if (perm.username == username &&
filepath.toCanonical().toString().startsWith(perm.filepath.toCanonical().toString
())) {
            return perm.write
        }
    }
    // 查找不到一律返回false
    return false
}
```

初始化FTP服务器时为特定的用户添加特定文件的权限，需要进行权限验证的场景例如上传和下载时，首先查找是否记录有该权限条目，存在则返回记录的权限，没有的默认返回无权限。