计算机网络LAB1-FTP实验文档

视频查看：

百度网盘

链接：https://pan.baidu.com/s/1vmV98yN-1XYQbnvuBt2iog
提取码：2eb0

贡献比：

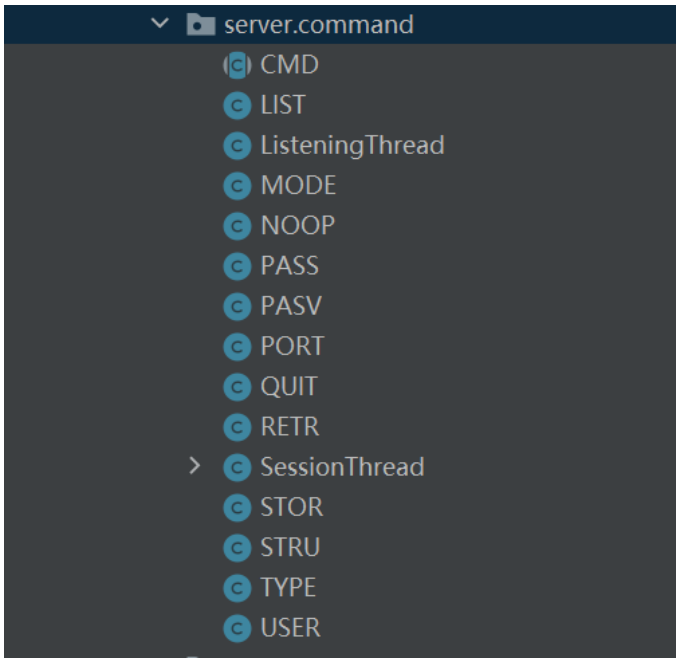
50/50

SERVER

实验人：18300120031周思瑜

command package：

目录结构如图：



CMD类：是实现了Runnable接口的抽象类。
解析控制连接上收到的命令，并运用java的反射机制获取各个命令类的构造方法来创建相应的对象，加强了程序的可扩展性。

```
CMD cmdInstance = null;
Class<?> clazz;
try {
    clazz = Class.forName(String.format("server.command.%s", cmdType));
    //获得构造函数，第一个是命令类型，第二个是命令参数,第三个是sessionThread
    Constructor<?> constructor = clazz.getConstructor(String.class,
String.class, SessionThread.class);
    //用构造函数创建对象并返回
    cmdInstance = (CMD) constructor.newInstance(cmdType, cmdArg, session);

} catch (Throwable e) {
    Log.e(TAG, "Instance creation error");
}
```

然后运用getClass()方法来拦截未登录用户，若已经登录则调用对应实例的run()方法

```
if (session.isLogin()||session.isAnonymous()) {
    cmdInstance.run();
} else if (cmdInstance.getClass().equals(USER.class)
        || cmdInstance.getClass().equals(PASS.class)
        || cmdInstance.getClass().equals(QUIT.class)) {
    cmdInstance.run();
} else {
    session.writeCmdResponse("530 Login first with USER and PASS, or QUIT");
}
```

SessionThread类：主要用于存放本次连接的一些信息和读取控制连接上的命令

ListeningThread类：监听用户的控制连接请求并accept

命令具体实现类：

USER

验证用户输入的username是否合法

PASS

验证用户名是否和密码相匹配（具体信息存储在FTPInfos中）

PORT

将FTP设置为主动模式（客户端打开端口），根据约定的接口文档将命令分解成一堆数字并按规律拼接成客户端的IP地址和端口号后通过set方法存储到sessionThread中

PASV

将FTP设置为被动模式（服务器打开端口），根据约定的接口文档解析命令并在服务器端新建并监听一个数据端口，用控制连接返回含有服务器ip地址和端口号的response

TYPE

通过if-else更改sessionThread中记录的 ASCII/BINARY 模式信息

MODE

通过switch-case更改sessionThread中记录的 Stream(流)/Block(块)/Compress(压缩) 模式信息

QUIT

重置sessionThread中的用户登录信息并关闭控制连接

LIST

用递归的方式将传入的文件夹中所有文件的文件名放入List中（用于上传和下载文件夹时循环发送命令）

RETR

若为主动模式，则主动连接之前客户端发送的ip和port（存放在sessionThread中），得到数据连接；

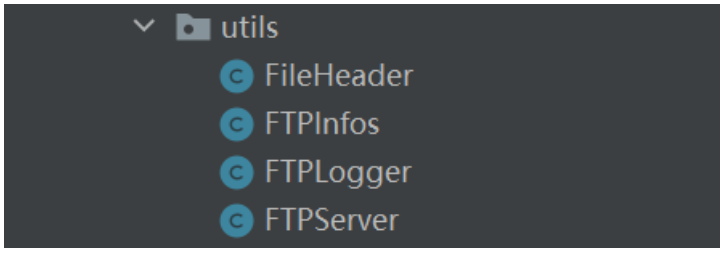
若为被动模式，则通过监听指定的端口accept得到一个数据连接（具体读写方式根据ASCII和BINARY的选择而变动）

STOR

客户端在数据连接上写入，服务端读取，其余部分与RETR类似

utils package：

目录结构如图：



FileHeader类：

用于存储文件的大小和文件名

FileInfos类：

用于存储FTP系统内置的用户名和密码对应信息以及FTP的存储根目录路径

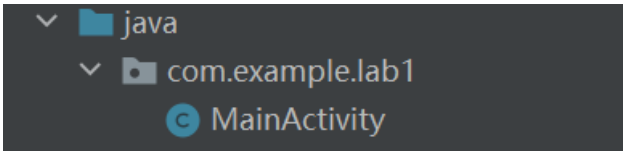
FTPLogger类：

用于连接logger和Android的UI界面

FTPServer类：

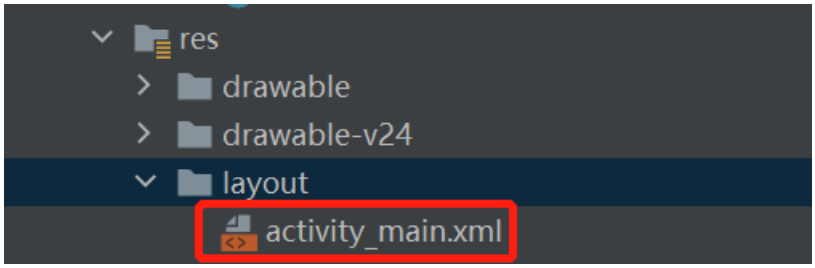
用于新建一个FTPServer并存储传入的server socket和thread

启动类：



主要通过lamda表达式设置各种UI控件的操作事件，且由于客户端要连接时需要服务器端IP，因此在界面最下端需要显示本机IP，由于Android4之后的版本都不允许在主线程中请求网络操作，因此从网上誊抄了一个从复杂网络环境中寻找IP地址的方法用于在主线程中调用

UI界面：



具体界面如下图所示，通过MainActivity类与逻辑部分相连接

CLIENT

实验人：19302010069 项心叶

所有核心代码都写在了`MyClient.java`这个文件夹里，其中包括所有指令的实现、文件传输方法的实现。这里简单介绍一下代码实现。

Exception

这里一共有两种`exception`，一种是`ClientException`，一种是`NoServerFoundException`。这些exception的产生除了一些代码上的exception（比如IOException）之外，还有服务器返回错误码时抛出的exception。

比如：

```
try {
    controlSocket = new Socket(host, port);
    controlSocketReader = new BufferedReader(new
InputStreamReader(controlSocket.getInputStream()));
    controlSocketWriter = new BufferedWriter(new
OutputStreamWriter(controlSocket.getOutputStream()));
} catch (IOException e) {
    throw new NoServerFoundException(e.getMessage());
 // 只是对已有exception做了一些包装
}
```

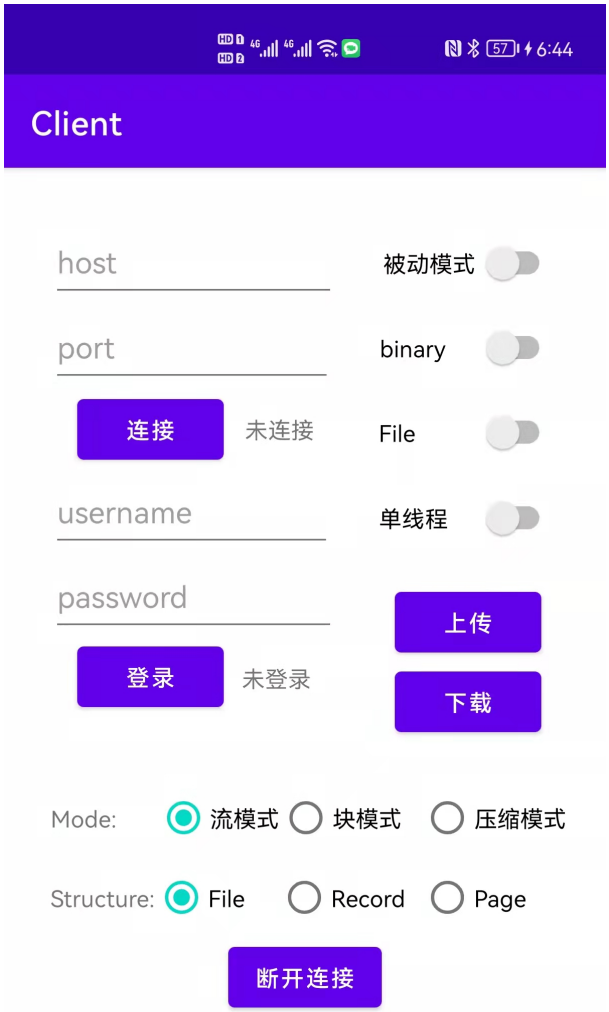
还有：

```
if (!resp.startsWith("230")) {
    throw new ClientException(resp);
 // 服务器没有返回成功码，抛错并把message设为服务器的回复
}
```

然后在前端提示用户时就直接显示message。

```
ToastUtil.showToast(MainActivity.this, e.getMessage(), Toast.LENGTH_SHORT);
// 显示错误信息
e.printStackTrace();
// 控制台打印错误相关信息，便于调试
```

UI界面：



优化

由于压缩算法对本次lab帮助不大，因此我们小组做的优化主要是针对文件夹的多线程传输。

对于普通的单线程文件夹传输，我们采用的主要是先用`LIST`指令列出文件夹下面所有文件（`RETR`的情况下），然后循环调用`STOR`指令或`RETR`指令传输文件夹中的文件。

主要的实现过程是，我们克隆了一个client对象，然后新开一个线程t，用原先线程传一半文件，用新开的线程再传一半，最后调用`t.join()`方法确保两个线程都传输完毕。

其中，克隆的代码如下：

```
public MyClient cloneClient() throws ClientException, NoServerFoundException {
        MyClient newClient = new MyClient(thisHost, thisPort);
        newClient.logIn(username, password);

        if (clientPattern == ClientPattern.PASSIVE) {
            newClient.pasv();
        } else {
            newClient.port();
```

```
        }

        newClient.type(typeOfCode);

        newClient.setDownloadDirectory(downloadDirectory);

        for (int i = 0; i < monitors.size(); i++) {
            newClient.addMonitor(monitors.get(i));
        }

        newClient.progressMonitor = progressMonitor;
        return newClient;
    }
```

并行执行的代码如下(以 `STOR` 为例):

```
        ArrayList<Exception> exceptions = new ArrayList<>();
Thread t = new Thread(() -> {
            try {
                for (int i = 0; i < absoluteFilePathsA.size(); i++) {
  // 这里的A表示前一半文件
                    client
```