本次的PJ我们分了两部分来实现：安卓UI部分和FTP部分

分数划分：1：1

### FTP部分：

**实现原理：**

ftp是TCP/IP协议的应用，因此ftp部分主要是通过java的socket接口开始实现的

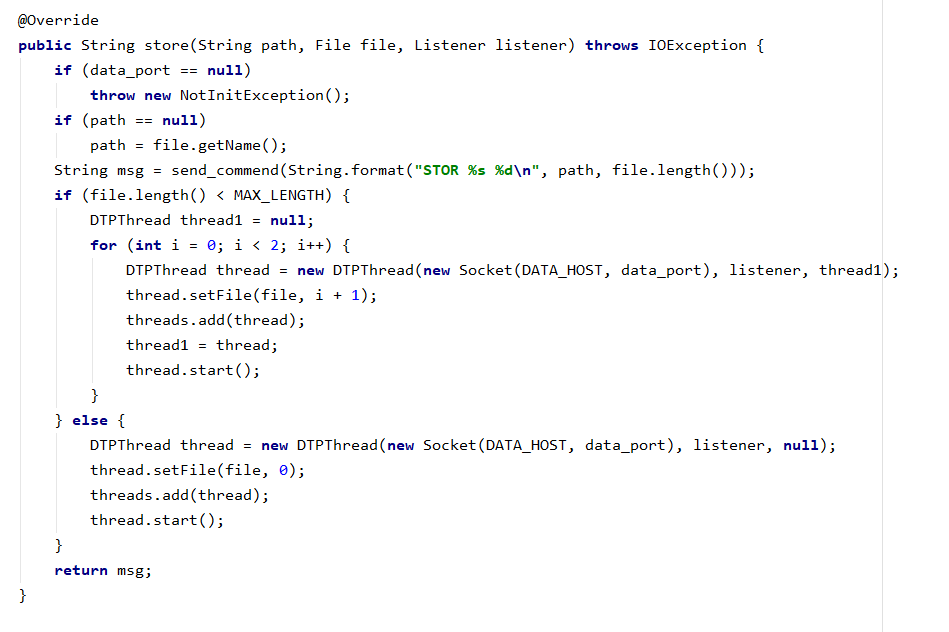
1. 服务端和客户端分别维持一个PISocket来保持命令的通信

2. 关于文件的的传输的指令都会新创建一个数据的TCP连接，即DTP，并用它来完成文件的传输

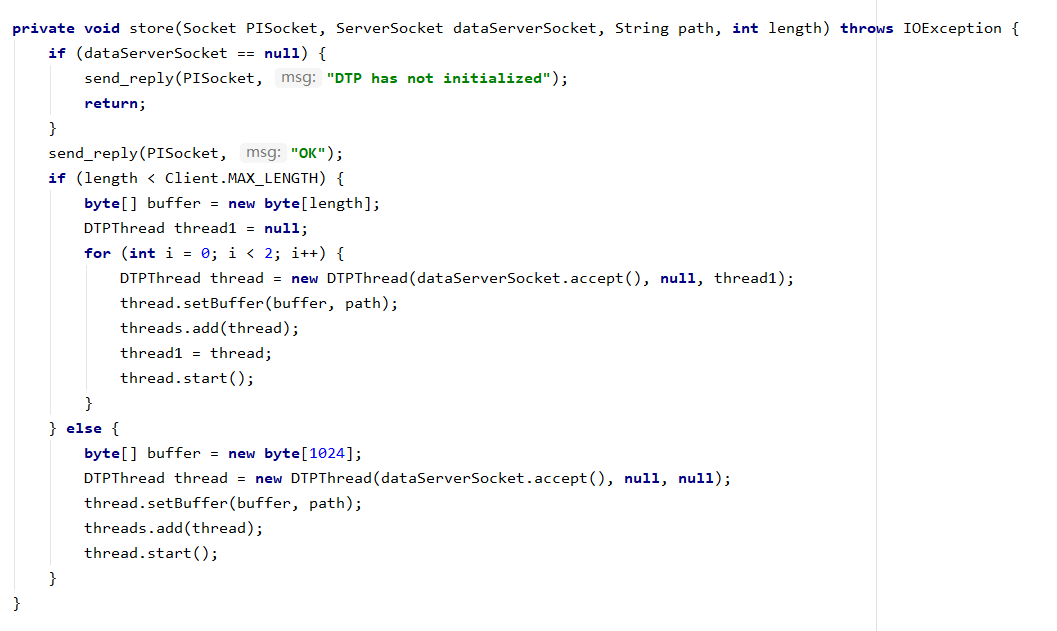
3. 因为安卓的限制，每个指令都要客户端新开一个线程的来进行操作，并且传输文件的操作并不知道何时会结束，因此相比其他指令还要多开一个线程，所以就需要创建Listener类来监听文件传输的状态

**文件传输部分详解（以STOR为例）：**

**客户端：**

****

**服务端：**

****

如图所示，客户端为首先给服务端发送传输指令，要求服务端开启数据传输接口，服务端发送许可回应，并开启相应接口监听客户端的连接请求。

当数据连接建立后，客户端和服务端都会另开一个线程来完成传输工作，这一部分是由DTPThread类来完成的。

**传输优化：**

传输优化主要是对小文件的传输做了优化。

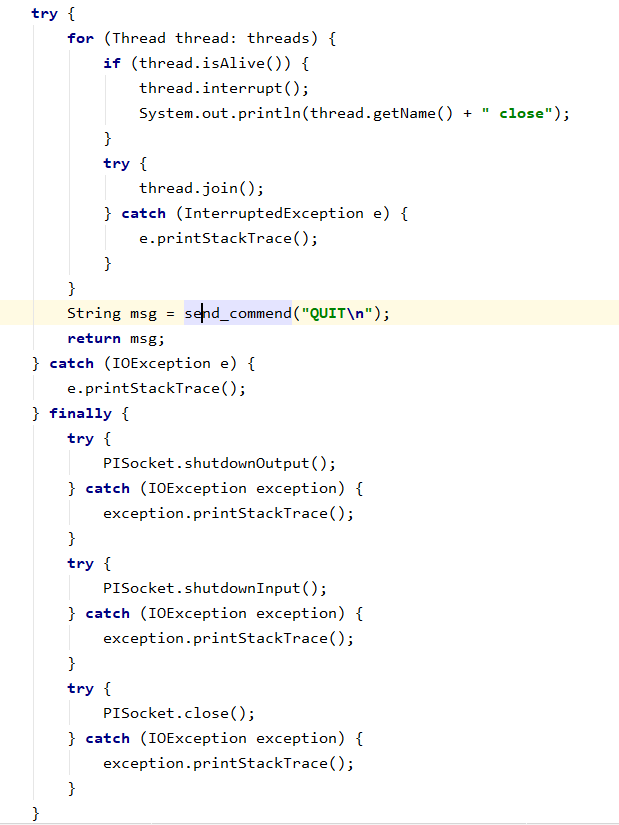
如上图所示，很明显可以看出当文件大小小于一个阈值时（100M）时，会开启两个线程来并行完成传输工作。

为了完成这一操作，发送方会预先发送文件大小，由接收方创建接收字节数组，因此若文件大小过大，可能出现接收字节数组过大超过内存极限的情况，所以只在小文件做了优化。除此之外，为确保接受的数据顺序正确，需要各线程额外发送自己负责的是文件的哪一部分，接收方会根据这一信息来安排接受的数据顺序。

**鲁棒性展示：**

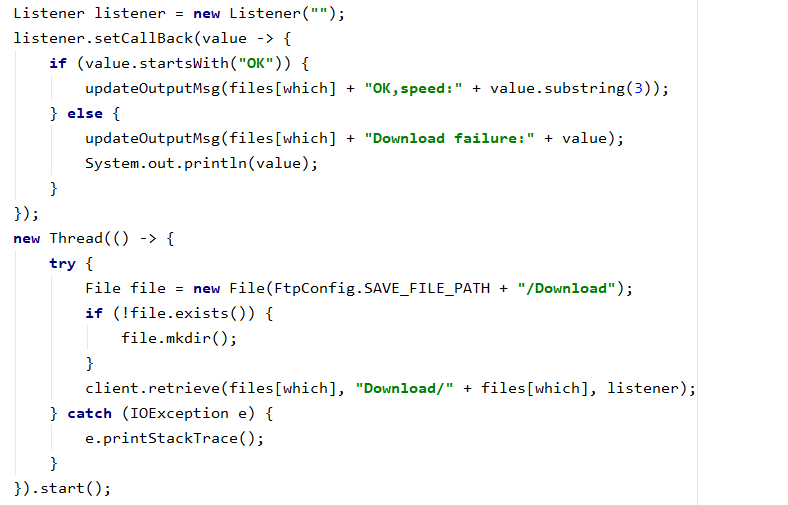
因为设计socket操作，大量的错误都有不确定性，因此为保证鲁棒性和socket，数据流等资源的不泄露，这些资源的关闭大都在finally部分完成

以客户端的quit操作为例：



会妥善的中断所有数据传输线程，然后再小心地释放资源，确保鲁棒性

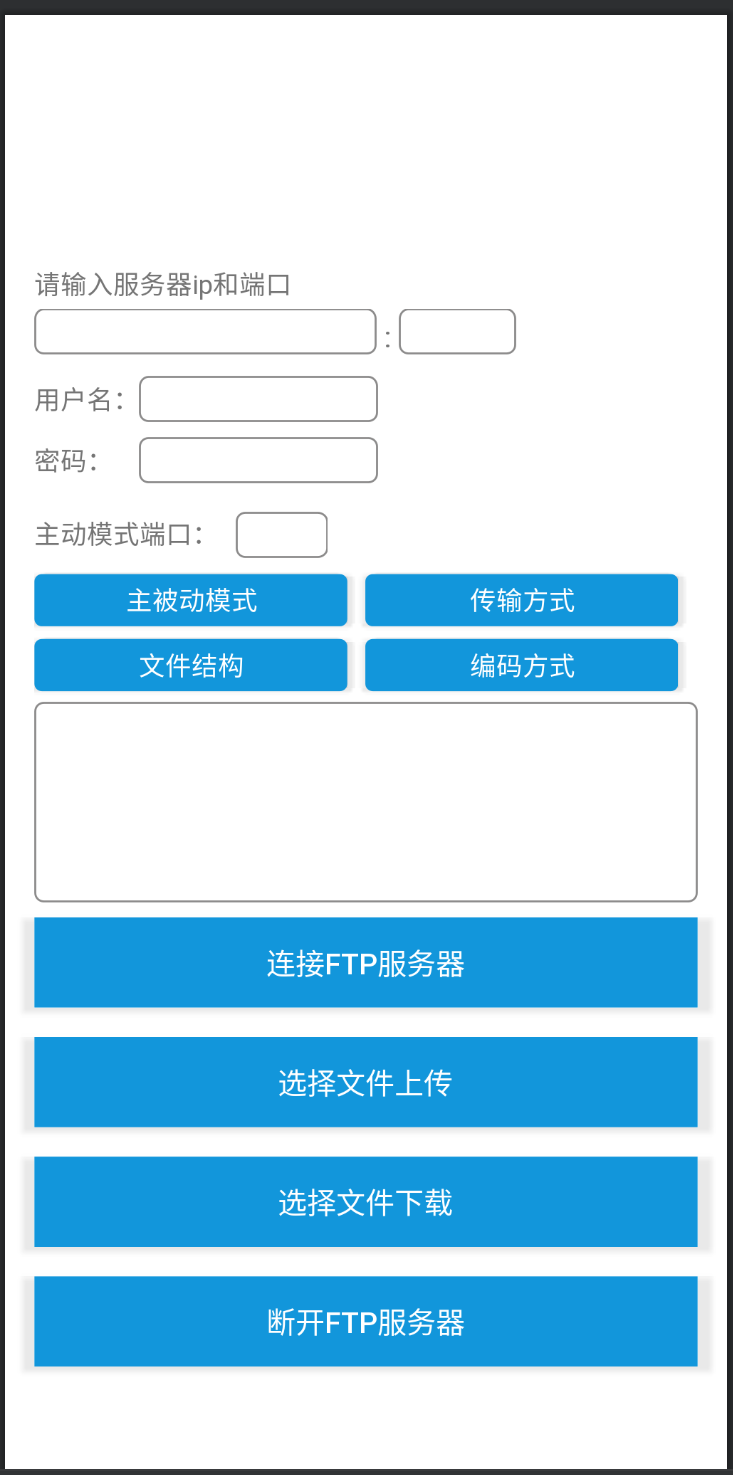
除此之外，因文件的传输是在后台进行的，所以加入了监听器确保当文件传输失败时会通知用户：



### Android部分

本项目安卓部分UI界面相对简单

主要活动都在ClientActivity和ServerActivitiy这两个界面中展开：



遇到的问题：

1. 关于权限的获取（尤其是对目前已普及的安卓11版本手机）：

在安卓11中由于对于隐私权限的全方面禁止，导致绝大多数的时候，即使申请了权限，也不能对文件进行有效读写，对项目进度造成了极大困扰

1. 安卓的主线程（UI）与多线程之间的关系充分分离了前后端之间的联系，为传输优化带来了一些障碍，并且安卓系统为每个线程分配的内存都比较少，有时很难在不变动app参数的情况下达成期望的设计，需要谨慎使用内存空间。我们也在此处遇到了问题，同通过增加文件流的写入大小限制，避免出现OOM。
2. 有关文件读写的操作中，安卓并不允许在还未生成文件夹的目录下进行文件读写操作，这个小技巧导致了很多的权限错误，需要先makedir在读写文件。
3. 在打开程序之前进行动态权限申请可以保证程序的正确运行，避免出现未知的bug。
4. 在调取后端代码时，通常新开线程避免主线程错误，常见于按钮点击事件，如果需要在非主线程中修改UI内容需要使用runOnUiThread进行进程切换。举例如下：



