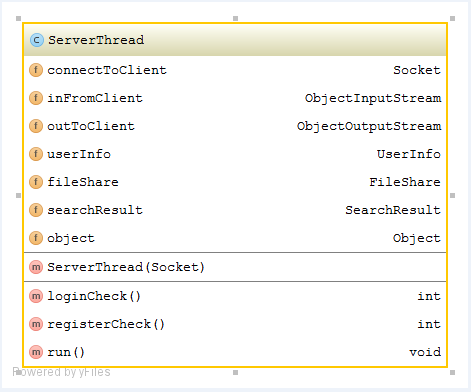
# 开发说明文档

1. 服务端实现说明

本系统服务器端业务处理部分是一个继承了Thread的线程类ServerThread。该类有以下成员变量：

1. private Socket connectToClient; 该对象用于建立与客户端的链接
2. private ObjectInputStream inFromClient;该对象用于接收从客户端传来的对象
3. private ObjectOutputStream outToClient;该对象用于向客户端发送对象。
4. private FileShare fileshare;该对象用于存储一条完整的文件信息，包括文件名，文件路径，上传的用户名，端口号等等。
5. private SearchResult searchResult;该对象用于存储服务器查询数据库返回的结果链表，链表里存放的对象类型是FileShare。
6. private Objecet object;该对象用于暂存从客户端接收到的对象，并进行对象所属类型的判别，例如：if(object instanceof FileShare)以此来处理不同对象的业务。
7. private UserInfo userInfo;该对象用以向数据库注册用户信息或者登陆时检查用户信息是否正确。

该类的UML图如下图所示：UML图由Intellij IDEA编译器自动生成



在构造函数中会建立与客户端的连接，并且实例化inFromClient与outToClient，用

以在客户端与服务器之间传递对象。

loginCheck()与registerCheck()函数均在run()函数中调用，用来实现登陆以及注册功能。

服务器主要接收从客户端传来的各种对象，并通过对不同对象类型的判别，以及对象内部的flag标记来实现不同的功能。服务器也是与数据库交互的唯一接口，用于查询更新数据库的内容，并把所得的内容放入存储信息的对象中，如UserInfo类和FileShare类及SearchResult类。通过这些类的对象实现在客户端和服务期间交换数据信息。本系统还有一个专门用于服务器向客户端返回执行状态的ResultFlag类，该类存储了一个int型变量，客户端可以根据此数值得知某个操作是否在服务器端操作成功，例如：当服务器返回的resultFlag对象内的Int值为1即告诉客户端用户注册成功，2表示用户注册失败等等共7个状态。

本系统服务器即ServerThread线程类只处理除传输文件之外的功能：用户注册，登录，查询文件，上传文件，显示文件列表。下载部分在由TransportServer类在客户端之间实现。

服务器启动在Server类中，该部分代码很简单如下：

**public class** Server {  
 **public static void** main(String[] args) **throws** IOException {  
 System.***out***.println(**"建立并等待连接……"**);  
 ServerSocket serverSocket = **new** ServerSocket(5500); *//指定服务器要监听的端口* Socket connectToClient = **null**;  
 **while** (**true**) {  
 connectToClient = serverSocket.accept(); *//监听5500端口* **new** ServerThread(connectToClient);  
 }  
 }  
 }

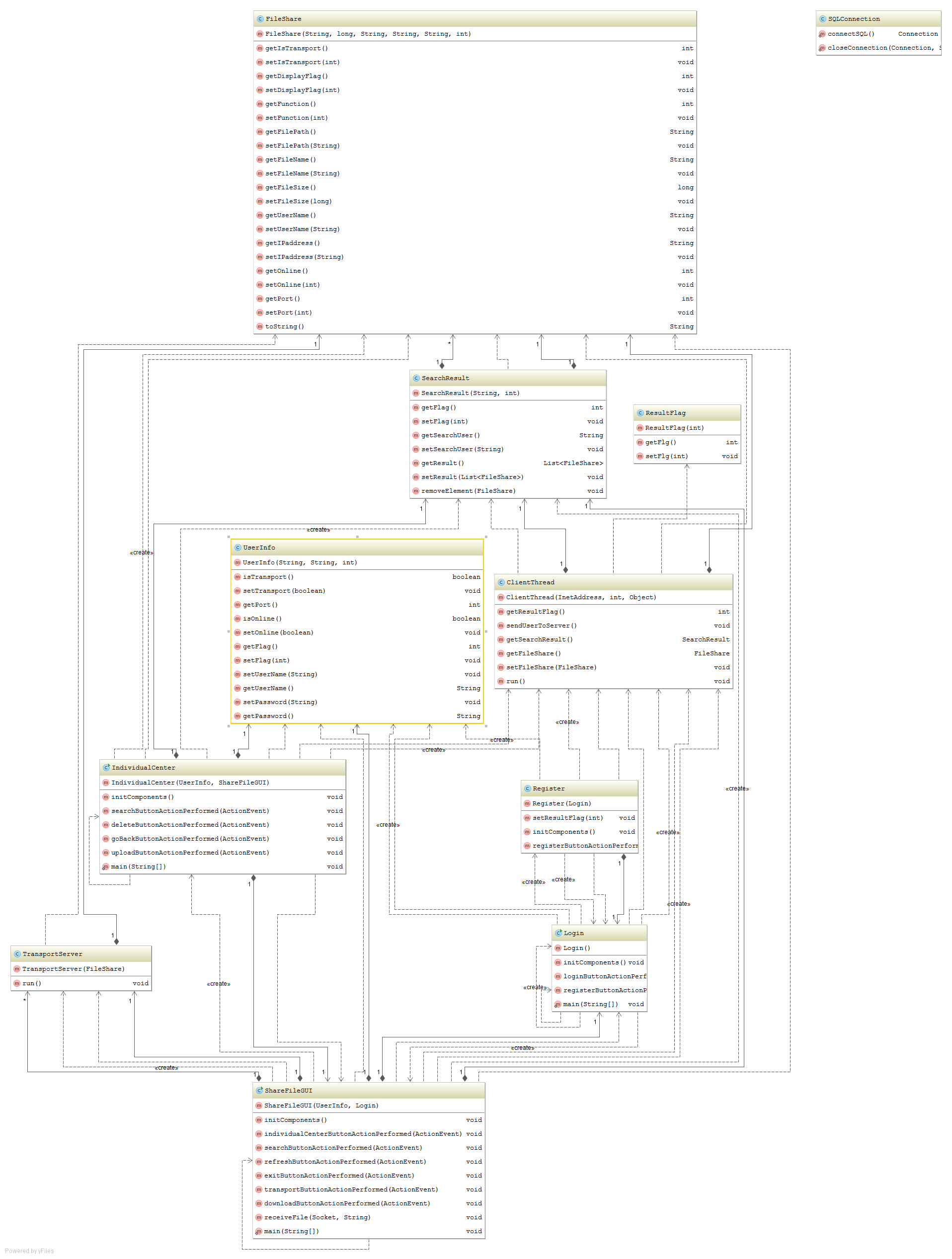
服务器线程只用一个端口5500来监听所有客户端发来的请求（不包括下载任务）

每监听到一个请求就开启一个新的服务器线程处理申请的任务。

1. 客户端实现说明

本系统由于是一个P2P系统，因此客户端在有用户要下载本地的文件时，同时充

当服务器的角色。客户端UML图如下：UML图由Intellij IDEA编译器自动生成。



由于客户端需要很多图形化界面，因此本系统客户端界面均使用netbeans编译器直接拖拉swing控件生成，因此自动生成的代码比较相似。

客户端启动界面即Login类，在该类中得到用户输入的用户名和密码，并把用户名和密码写入UserInfo类的对象中，并传递给服务器用于在数据库中检索是否存在该用户，若存在则判断用户名与密码是否相匹配，并将结果状态返回给客户端，客户端可以根据此状态信息向用户提示。对用户名、密码空输入的判断并没有交由服务器判断，而是在客户端本地进行判断。该类中有两个Label标签，默认值不显示，只有当用户名或密码为空时，才会在界面显示，以提醒用户输入非空的用户名和密码。Login界面有一个“注册”按钮，当点击此按钮后会进入注册界面（Register类），该类实现的功能原理与login类似。

用户登录成功后会将记录该用户信息的UserInfo类一同传递给后续界面。在ShareFileGUI界面中，用户可以在此查看到所有数据库中已有的文件信息，也可以查找想要的文件。文件的显示使用JList来实现。当用户点击“刷新”按钮后，会有一条请求发送给服务器，该请求信息存储在SearchRuselt类中，并且在其中设置标记为3表示此次处理用于查询数据库中所有文件信息。服务器端根据对象的种类和标记得知具体的任务，例如此次就会将所有数据库的文件信息存储在SearchResul的链表中，再将此类对象返回给客户端，客户端从其中的链表中读出各条文件信息，并显示在客户端的JList中。点击“个人中心”可以进到用户管理文件界面（IndividualCenter），在此界面也有一个JList列表，不过此列表中的信息仅有该用户上传的文件。得到文件列表的方法和前文相同，只不过标记设为1表示按用户名搜索文件。在此界面中可以上传文件，上传的文件信息存储在FileShare类的对象中，服务器接收到此对象后会检索文件路径名在数据库中是否已存在，若有重名，则返回状态5表示该文件已存在，反之，则将文件信息写入数据库，每个文件的端口号均不相同，从2000开始逐个加1。在“个人中心”界面还可以删除该用户上传的某个文件，方法与上传类似，实质是在数据库中把符合条件的记录删除。

文件的下载功能在前面提到的“ShareFileGUI”界面中实现。该类中有一个存储了TransportServer对象的链表。当点击“传输开/关”按钮后，会依次取得该用户上传的每个文件的端口号，并将此端口号传给一个新的TransportServer线程，用于监听相应的端口，并且同时更新数据库中此用户isTransport状态，告知其他用户此用户的文件允许下载。传输开启后，会遍历线程链表，并逐个调用start()方法，启动监听线程，若有其他用户请求某端口的文件下载，则传输文件。当传输关闭后，会同时更新数据库中isTransport信息以及遍历此该线程列表，逐个调用stop()方法关闭监听线程。

当用户点击JList列表中的文件同时点击“下载文件”按钮后，会从选中的文件（FileShare）中得到该文件的端口号和路径名，并将两个信息作为参数传入receiveFile函数中，并建立对应端口的Socket请求，如果此文件上传用户允许下载，则会建立连接进行下载，否则会返回提示“此文件不允许下载”信息。

1. 运行结果截图和说明

本系统服务器端并没有图形化界面，只在命令行中显示关键的提示信息。当服务器启动后处于监听状态时的信息显示如图1-1所示。

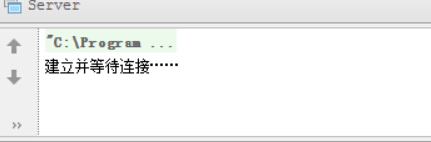


图1-1

当前数据库userinfo表无记录，如图1-2所示。

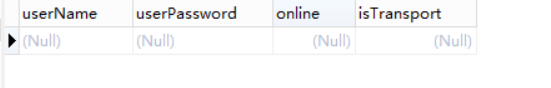


图1-2

此时启动客户端界面，如图1-3所示。

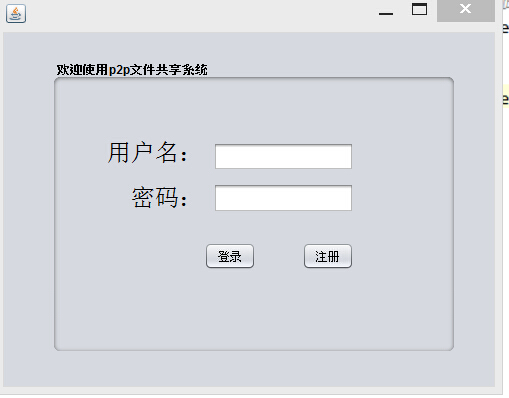


图1-3

当用户名密码输入为空时，提示状态如图1-4所示。

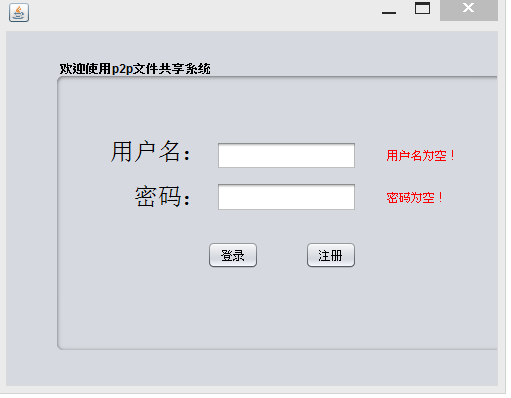


图1-4

点击注册会进入用户注册界面，如图1-5所示。



图1-5

若输入为空，情况如图1-6所示。



图1-6

输入正确的用户名与密码，点击注册后，如图1-7所示。



图1-7

此时数据库userinfo表的状态，如图1-8所示。

C:\Users\wk_51920\AppData\Roaming\Tencent\Users\307780762\QQ\WinTemp\RichOle\V[Q@B5CLVPOK}G95UQ_]7ZY.jpg

图1-8

此时若还使用此用户名注册，则会如图1-9所示。



图1-9

此时若使用一个为注册的用户名登录系统或者用户名密码不匹配，会有如下提示，如图1-10。



图1-10

现在登录账户，登录成功后的的页面如图1-11所示。



图1-11

此时数据库userinfo表状态如图1-12所示。

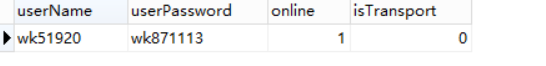


图1-12

Online表示此用户在线，已登录。如果此时在另一个客户端使用此用户名登录，则如图1-13所示。

图1-13

进入“个人中心”先上传文件，“个人中心”界面如图1-14所示。

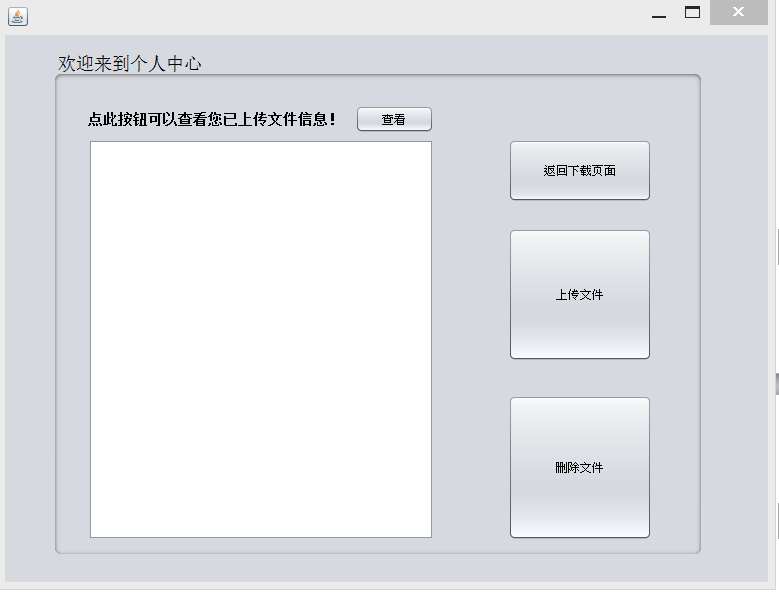


图1-14

此时数据库中fileshare表中状态如图1-15所示。

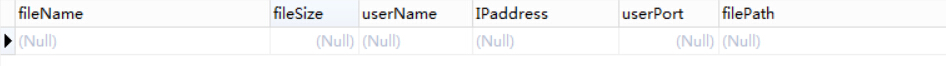


图1-15

此时无任何文件记录，点击“上传文件”按钮后，选择要上传的文件，如图1-16。

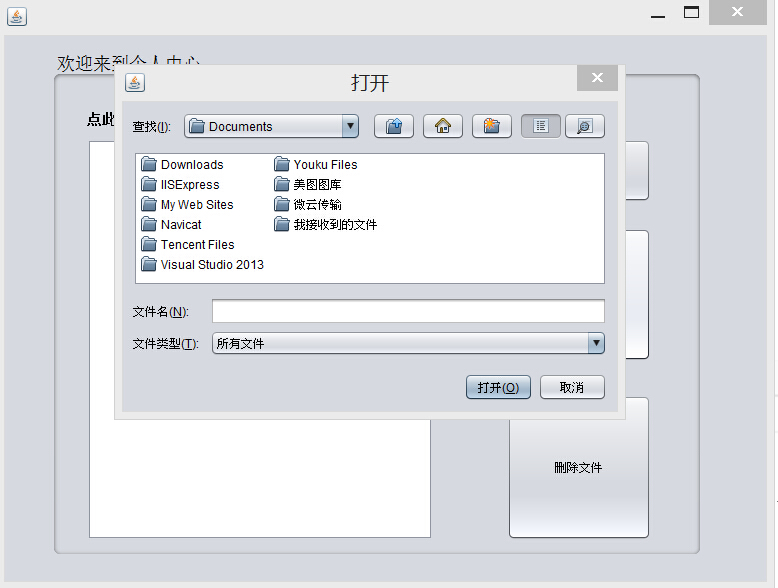


图1-16

本次上传一个三个文件(文件名必须是英文名)，成功后的提示信息如图1-17所示。



图1-17

此时数据库fileshare表的状态如图1-18所示。



图1-18

点击“查看”按钮可以查看所有此用户上传的文件，如图1-19所示。

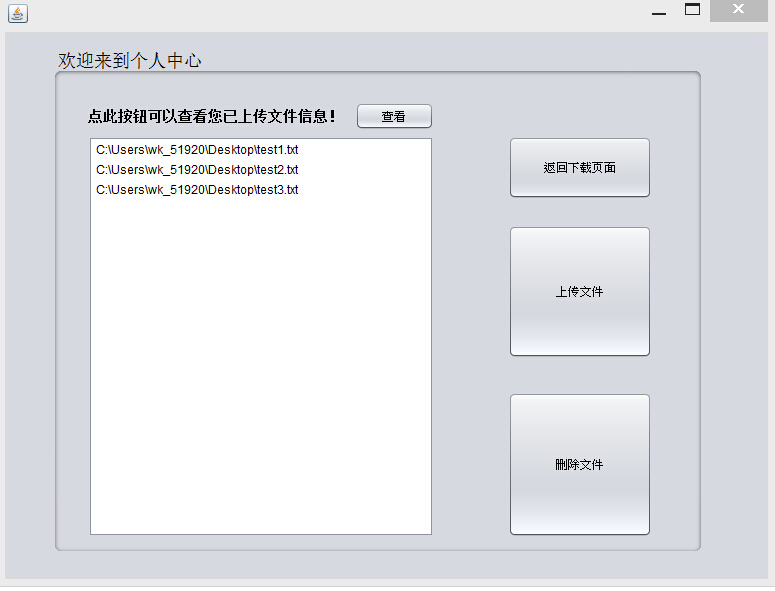


图1-19

选中test2,并点击“删除文件”按钮，如图1-20所示。

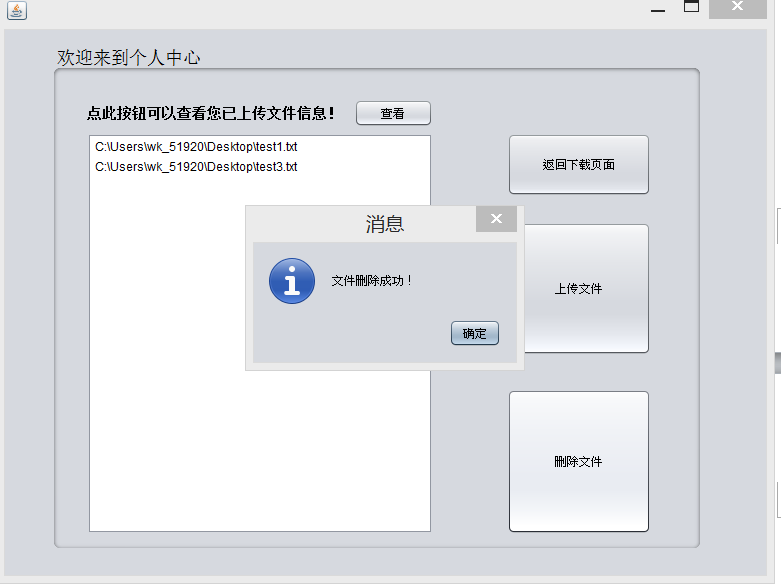


图1-20

此时数据中fileshare表状态如图1-21所示。



图1-21

点击“返回下载页面”返回之前页面。点击“刷新”按钮，如图1-22所示。

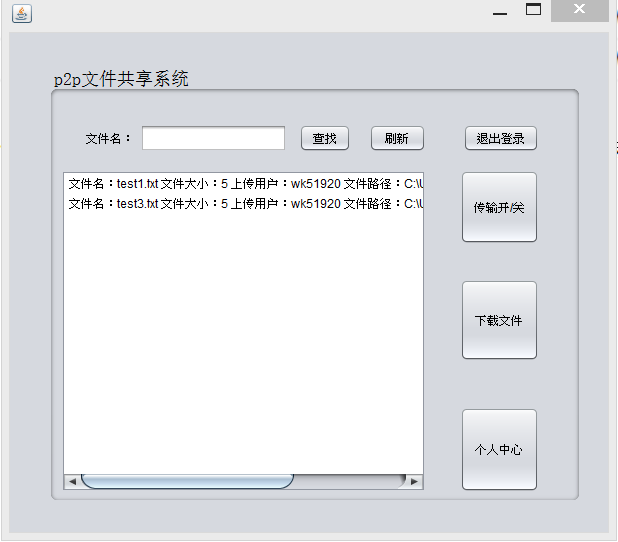


图1-22

如果使用查找功能，则输入想要查找的文件名，结果如图1-23所示。

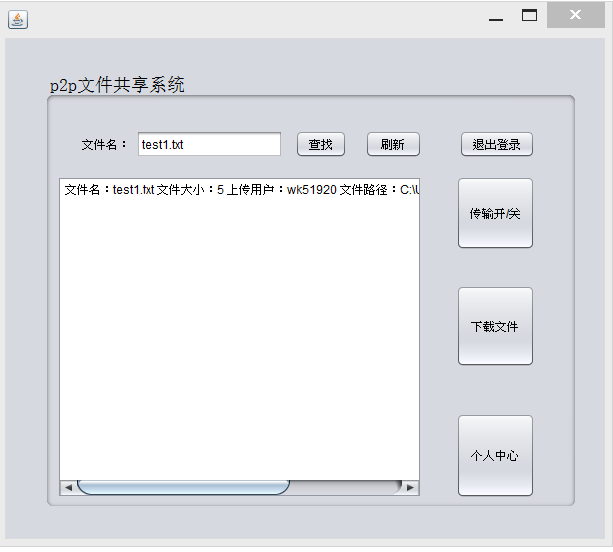


图1-23

如果允许传输本用户上传的所有文件，则点击“传输开/关”，结果如图1-24所示。

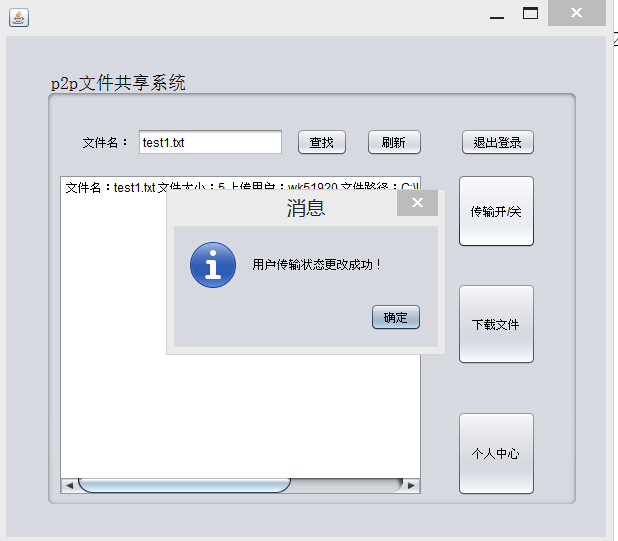


图1-24

此时数据库中userinfo表状态如图1-25所示。



图1-25

isTransport为1表示该用户的文件允许下载，且所有该用户上传的文件均打开了一个监听线程，当有此文件端口的请求到来时，建立连接并进行传输。

此时选择列表中的一个文件，点击“下载文件”按钮，弹出选择存储路径对话框，选择文件存储位置，如图1-26所示。

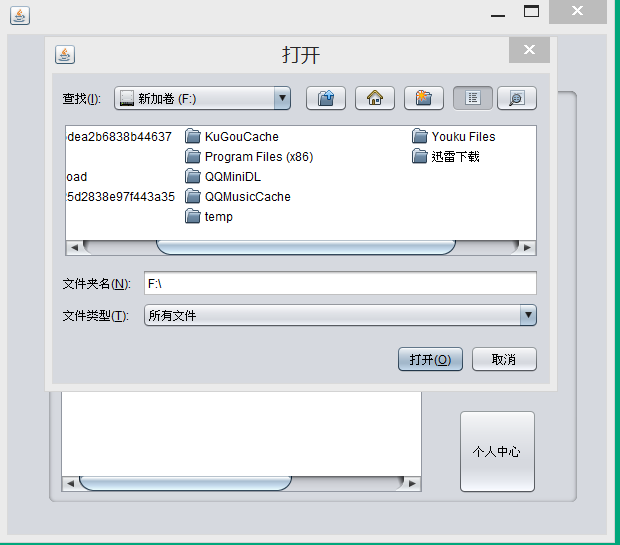


图1-26

将文件test1.txt存入F盘根目录下，传输过程在控制台显示如图1-27所示。

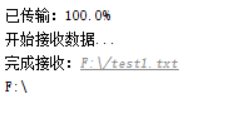


图1-27

此时文件已传输至F盘根目录下。此时关闭传输功能，如图1-28所示。

C:\Users\wk_51920\AppData\Roaming\Tencent\Users\307780762\QQ\WinTemp\RichOle\EBSYZYMDXEMDB%QU@)C0~7J.png

图1-28

再选择test1.txt进行下载，则如图1-29所示。

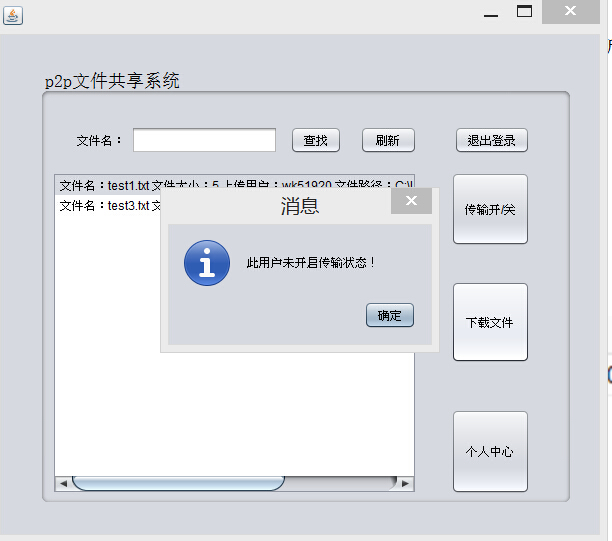


图1-29

现在退出系统，数据库userinfo表状态如图1-30所示。



图1-30

online为0，表示用户确实已经下线，在退出登录时，会同时把该用户的online以及isTransport状态置为0。

1. 开发过程遇到的问题

本人由于之前并没有客户端网络编程的经验，因此在系统开发过程中遇到了大大小小的问题。在此只写出以下几个我认为比较重要的问题。

1. 数据库状态和用户行为一致性问题。

本系统很多关键动作依赖于数据库中userinfo表中的online和isTransport状

态信息。最开始写程序时并没有意识到此问题，直到有一次从FileShareGUI界面退出到登陆界面的时候点击的是右上角的叉按钮，而不是界面中的“退出登录”按钮。此时发现数据库中的online信息和isTransport信息并没有变更，才意识到有漏洞的存在。虽然设置了专门的“退出登录”按钮，但有些用户会习惯性地通过点击关闭按钮来退出，如果不对关闭按钮进行监听，就会造成用户退出了系统，当数据库却显示还未退出的问题。本系统每个界面的关闭按钮都有相应的监听事件来保证系统状态一致。

1. Windows文件路径反斜杠转义问题

Windows的路径名中各级之间使用反斜杠连接，但反斜杠在很多场合都有转

义的语义。如果没有认识到这一点，那么当将文件路径写进数据库的时候就会出现问题。刚开始一直纠结于为何在控制台可以输出完整路径，而在数据库中路径中所有反斜杠均消失了。通过在网上查找相关问题才找到了问题的根源。在JAVA中反斜杠同样有转义的语义，所以写语句时还要考虑到双重转义。

String convert = **fileShare**.getFilePath().replaceAll(**"\\\\"**, **"\\\\\\\\"**);

本人即使用上条语句对路径中的所有反斜杠进行转义，注意此处\\\\只代表路径中的一个反斜杠 。

1. 建立传输对象连接的问题

如果客户端在构造函数中要同时初始化

**inFromServer** = **new** ObjectInputStream(**connectToServer**.getInputStream()); **outToServer** = **new** ObjectOutputStream(**connectToServer**.getOutputStream()); 、

这两个传输流如果都为ObjectInptStream。初始化顺序如上语句。则在服务器端初始化这两个传输流时顺序必须为：

**outToClient** = **new** ObjectOutputStream(**connectToClient**.getOutputStream());  
**inFromClient** = **new** ObjectInputStream(**connectToClient**.getInputStream());

即：如果在客户端先初始化了发送对象的输出流，则在服务器端必须对应先初始化接收对象的输入流。如果这两条语句的初始化顺序出错，则程序直接死在那里。我在出现这个问题的时候调试了几个小时都调不通，因为没有系统的学习过网络编程，所以并不知道问题出现在初始化顺序这里，一直在别的地方不停地查找。最后也是在网络上有人专门指出了同时发送和接收对象时一定要注意正确的初始化顺序，否则程序直接死。