1 项目概况

1.1 项目简介

1.1.1 该课程项目是基于局域网的即时聊天系统，已实现功能：

1.1.1.1 对文本信息的处理以及正常传输和接收

1.1.1.1.1 涉及到的技术：

1.1.1.1.1.1 通过 UDP 的数据包传输

1.1.1.1.1.2 消息格式的定义

1.1.1.2 对语音信息的处理以及正常的传输和接收（语音通话）

1.1.1.2.1 通过 UDP 的数据包传输

1.1.1.2.2 通过 DataLine 的语音信息处理

1.1.1.3 对文件的处理以及正常的传输和接收

1.1.1.3.1 TCP 长连接的建立与停止

1.1.1.3.2 文件输入输出流的使用

1.1.1.4 刷新获取局域网计算机列表

1.1.1.4.1 按钮控件事件的设置

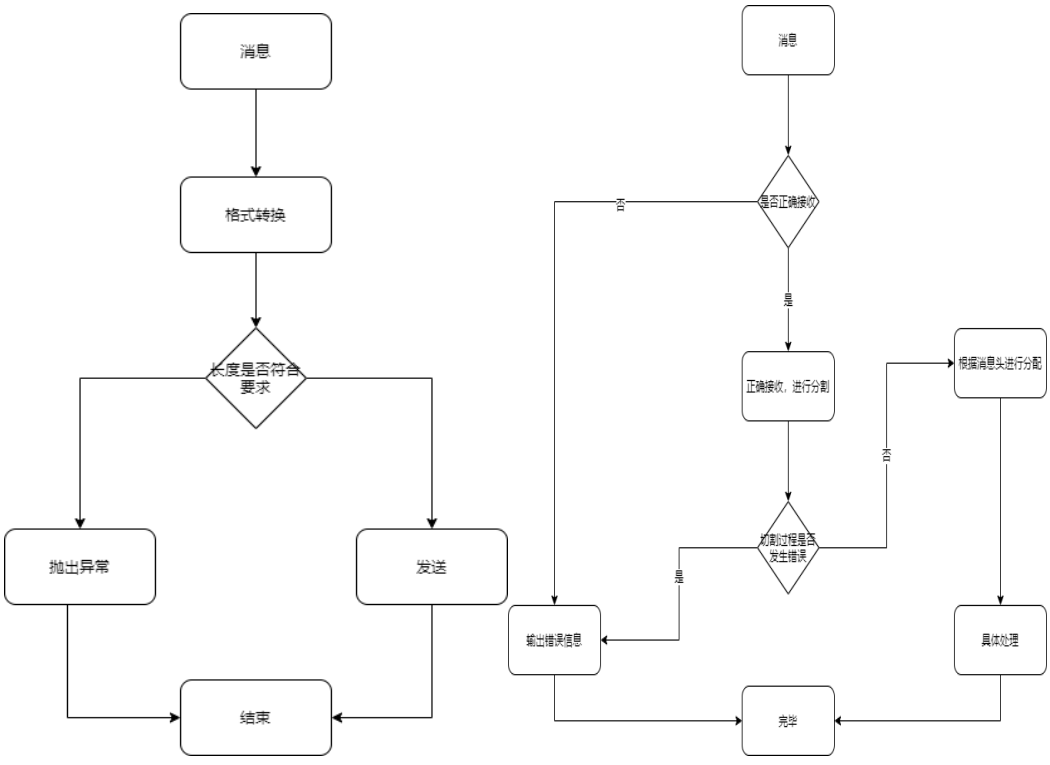
1.1.1.4.2 通过系统命令对计算机信息的获取

1.2 项目设计

1.2.1 项目业务分析(基于流程图的分析)

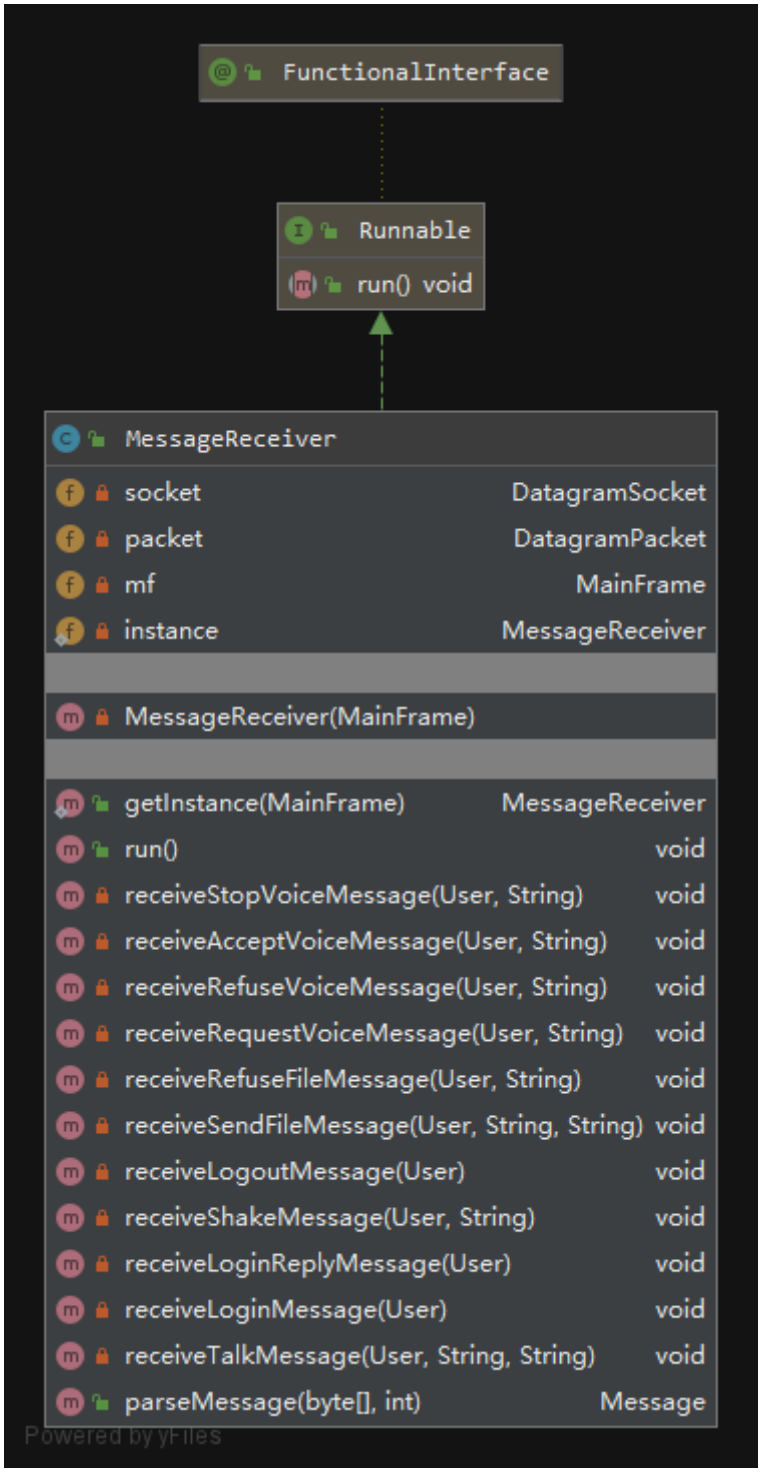
1.2.1.1 文本信息

1.2.1.1.1 控制流图

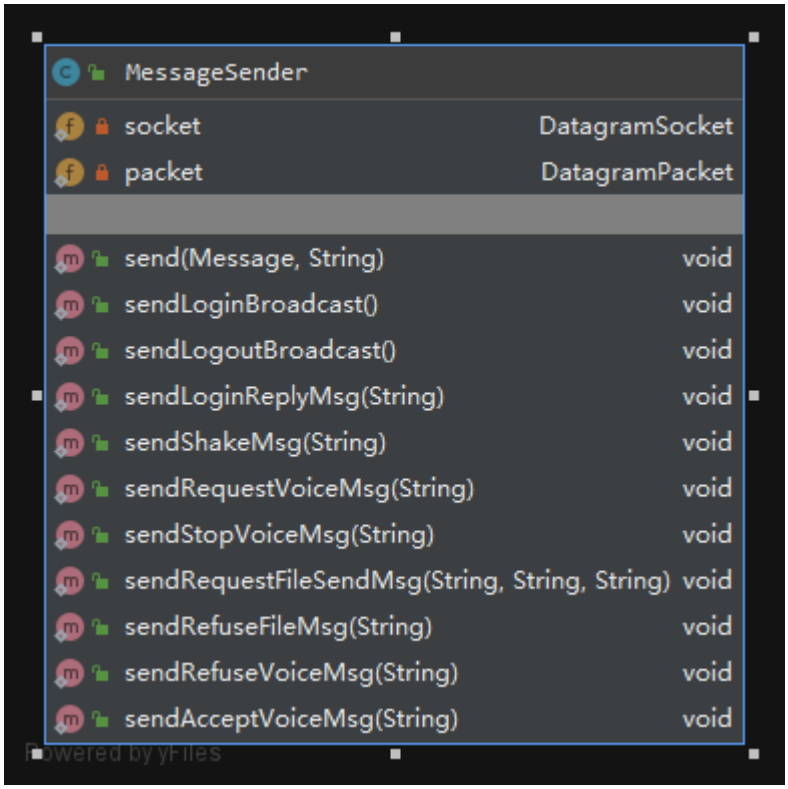


1.2.1.1.2 类图

1.2.1.1.2.1 MessageReceiver.java



1.2.1.1.2.2 MessageSender.java



1.2.1.1.3 类设计思路

1.2.1.1.3.1 MessageReceiver.java

1．一个用户只拥有一个消息队列，其他操作都是基于此基础上的线程操作，所以对该类采用单例模式，并采用懒汉式来保证线程安全；但是此方法通常在getInstance()的性能对应用程序不是很关键的时候，并且直接对方法加锁会浪费性能，所以改进项目拟采用 DCL 双检锁的形式改进，因为作为即时聊天系统，消息队列的调用肯定是频繁的。

```
public static synchronized MessageReceiver getInstance(MainFrame mf){
    if(instance == null){
        instance = new MessageReceiver(mf);
    }
    instance.mf = mf;
    return instance;
}
```

2．通过 udp 端口的传输获得消息，并根据消息不同的报头进行不同的反应，因为是根据参数不同进行不同处理，所以拟采用工厂模式进行改造。

```
if (msgHead.equals(Config.HEAD_TALK_MSG)) {
    receiveTalkMessage(user, info, msgData);
      UIUtils.playSound("sounds/msg.wav");
    //将信息封装写入数据库
    DBUtils.writeReceivedMsgToDB(msg, hostAddress);
} else if (msgHead.equals(Config.HEAD_LOGIN_MSG) && !isSelf) {
    // 登陆时发送的广播消息
    receiveLoginMessage(user);
}else if(msgHead.equals(Config.HEAD_LOGIN_REPLY_MSG)){
    // 回复登陆广播的消息，不用考虑是否为本机发送，因为本机收不到自身的登陆包
    receiveLoginReplyMessage(user);
}else if (msgHead.equals(Config.HEAD_LOGOUT_MSG) && !isSelf) {
    // 退出时发送的广播消息
    receiveLogoutMessage(user);
}else if(msgHead.equals(Config.HEAD_REQUEST_SEND_FILE)){
    //请求传送文件的消息
    receiveSendFileMessage(user, msgData, sendTime);
}else if(msgHead.equals(Config.HEAD_SHAKE)){
    //窗口抖动消息
    receiveShakeMessage(user, sendTime);
}else if(msgHead.equals(Config.HEAD_REFUSE_FILE)){
    //拒绝接收文件的消息
    receiveRefuseFileMessage(user, sendTime);
}else if(msgHead.equals(Config.HEAD_REQUEST_VOICE)){
    //请求语音聊天
    receiveRequestVoiceMessage(user,sendTime);
}else if(msgHead.equals(Config.HEAD_ACCEPT_VOICE)){
    //接受语音聊天
    receiveAcceptVoiceMessage(user,sendTime);
}else if(msgHead.equals(Config.HEAD_REFUSE_VOICE)){
    //拒绝语音聊天
    receiveRefuseVoiceMessage(user,sendTime);
}else if(msgHead.equals(Config.HEAD_STOP_VOICE)){
    //中断语音聊天
    receiveStopVoiceMessage(user,sendTime);
}
```

1.2.1.1.3.2 MessageSender.java

通过用户采取不同的动作来发送不同类型报头的消息

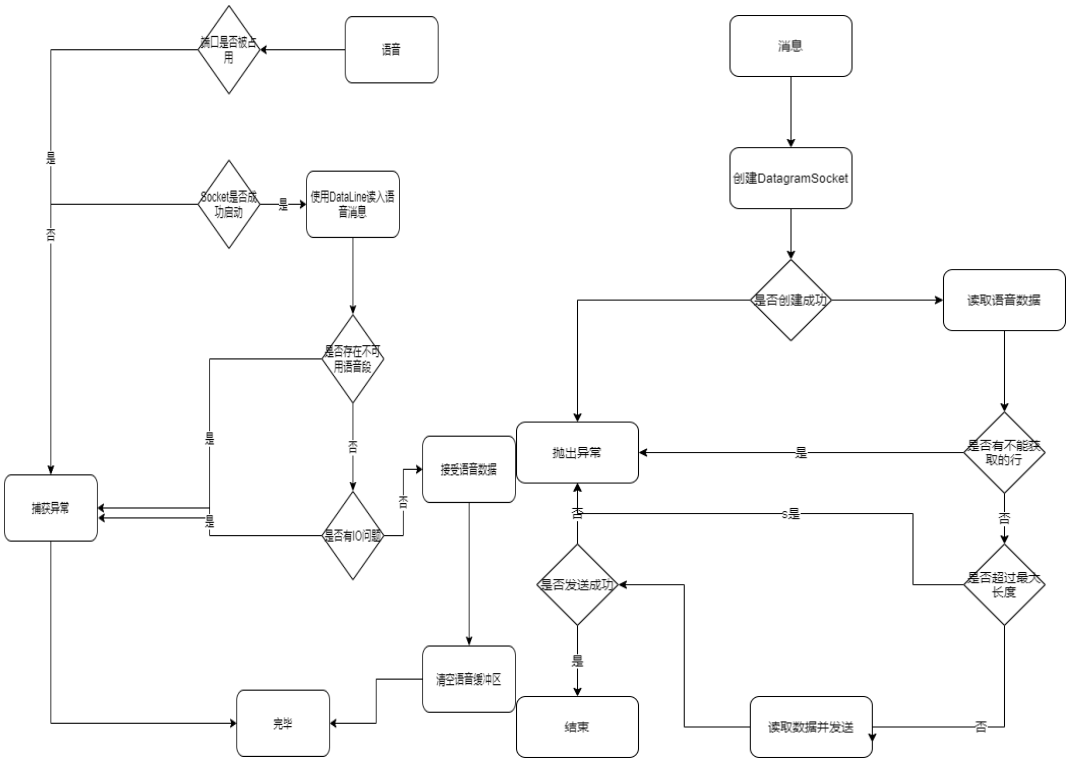
```java
public static void sendLogoutBroadcast(){
    //此处暂时取主机登录用户名
    String userName = SysUtils.getLoginUserName();
    Message msg = new Message(userName,Config.LOCAL_HOST_NAME,Config.HEAD_LOGOUT_M
    try {
        MessageSender.send(msg, Config.BROADCAST_ADDR);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void sendLoginReplyMsg(String hostAddress){
    //此处暂时取主机登录用户名
    String userName = SysUtils.getLoginUserName();
    Message replyMsg = new Message(userName,Config.LOCAL_HOST_NAME,Config.HEAD_LOG
    try {
        MessageSender.send(replyMsg, hostAddress);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void sendShakeMsg(String hostAddress){
    //此处暂时取主机登录用户名
    String userName = SysUtils.getLoginUserName();
    Message shakeMsg = new Message(userName,Config.LOCAL_HOST_NAME,Config.HEAD_SHA
    try {
        MessageSender.send(shakeMsg, hostAddress);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

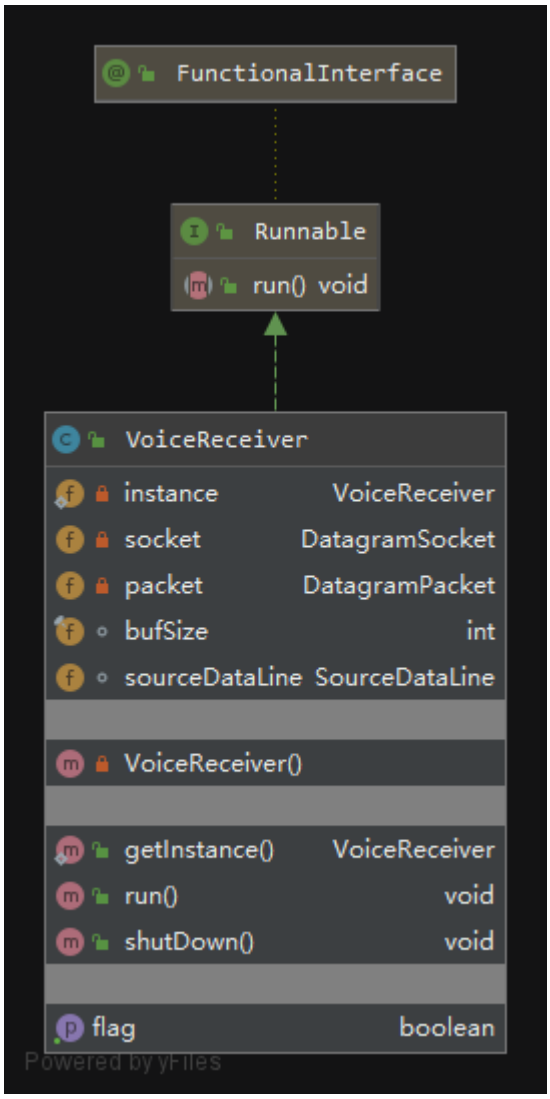
1.2.1.2 语音信息

1.2.1.2.1 控制流图

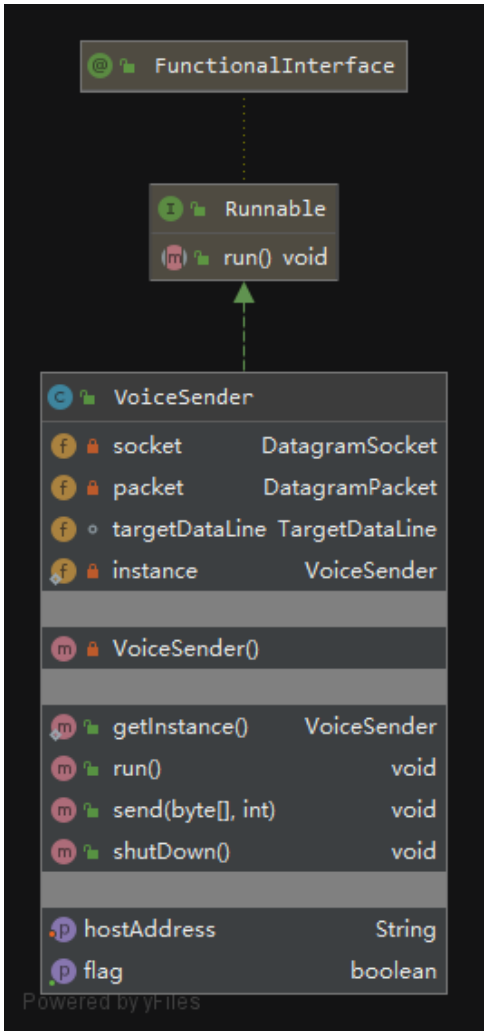


1.2.1.2.2 类图

1. VoiceReceiver.java



2. VoiceSender.java



1.2.1.2.3 类设计思路

1. VoiceReceiver.java

   一个用户同时只能拥有一个音频输出，所以采用单例模式，并且将会把懒汉式的单例改变成双检锁的单例

```
public static VoiceReceiver getInstance(){
    if(instance == null) {
        instance = new VoiceReceiver();
    }
    return instance;
}
```

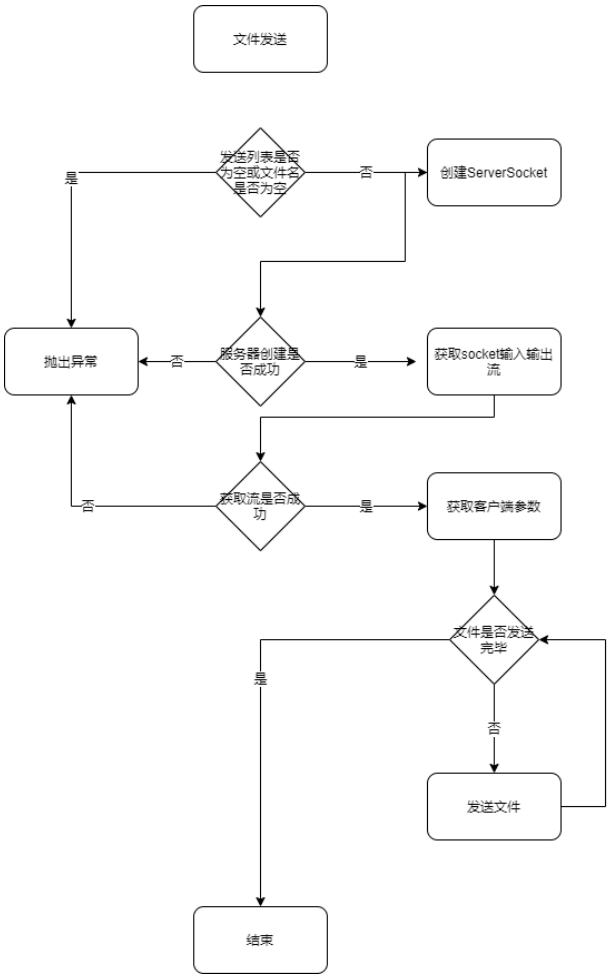
2. VoiceSender.java

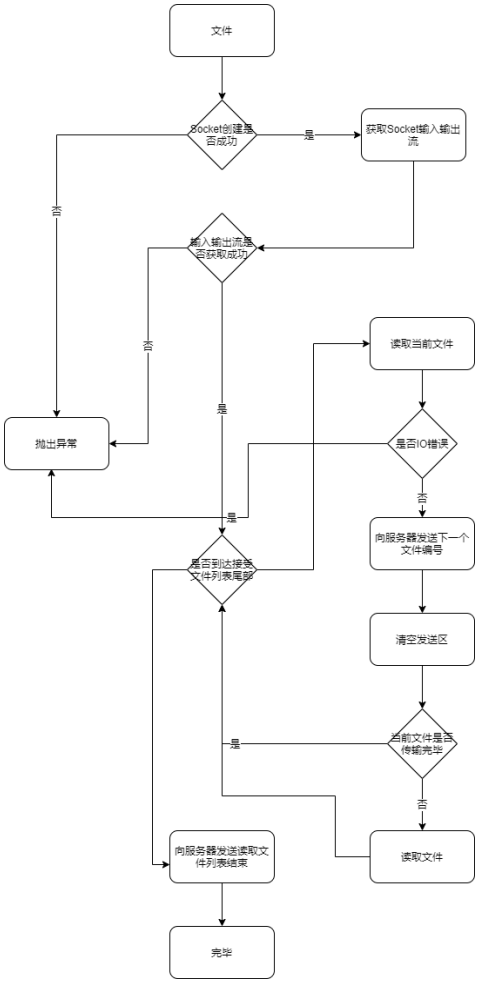
   一个用户同时只能拥有一个音频输入，所以采用单例模式，并且将会把懒汉式的单例改变成双检锁的单例

```
public static synchronized VoiceSender getInstance(){
    if(instance == null) {
        instance = new VoiceSender();
    }
    return instance;
}
```

1.2.1.3 文件传输

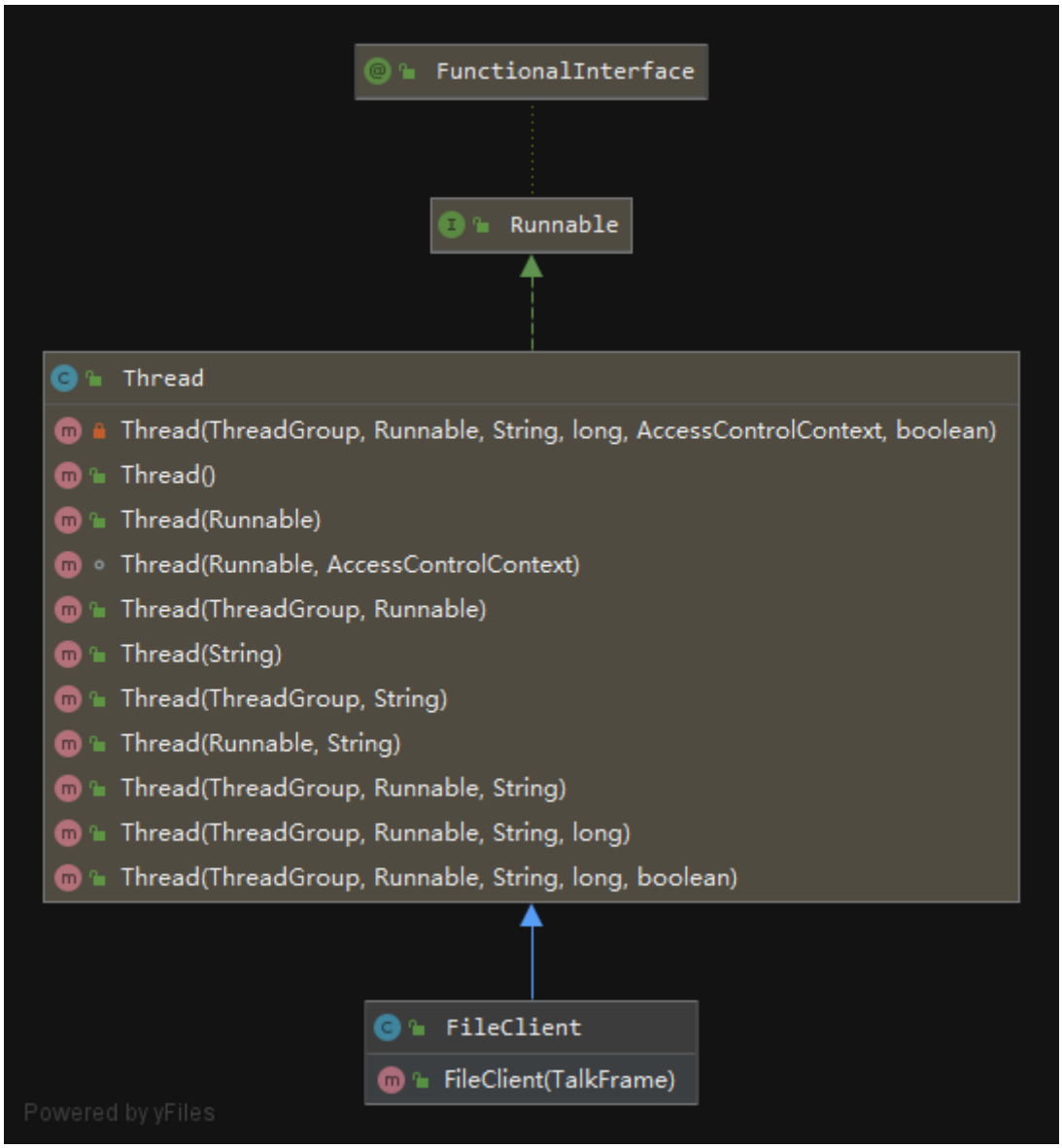
1.2.1.3.1 控制流图

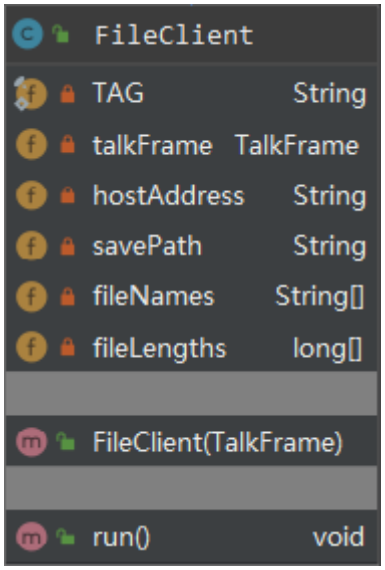




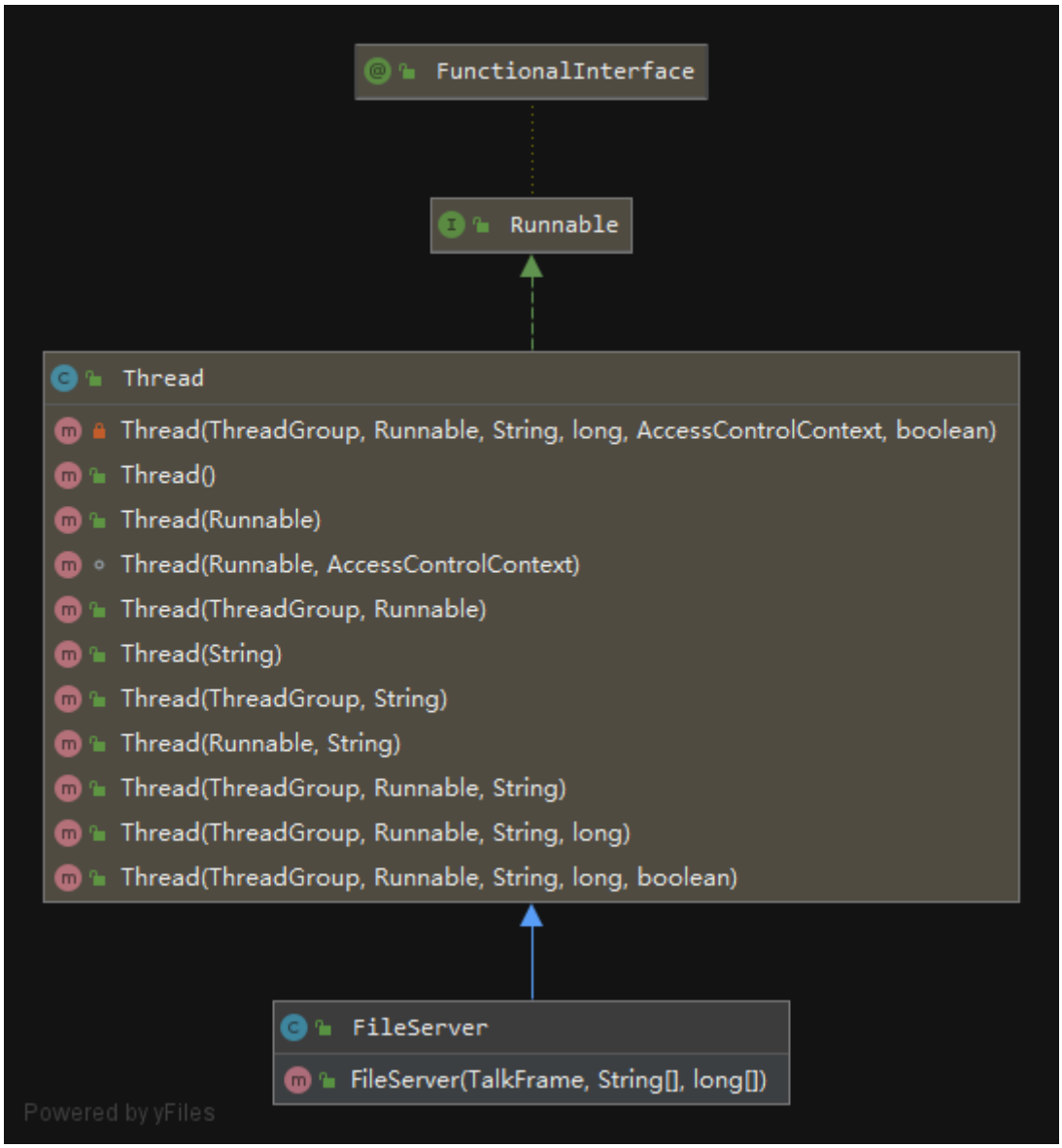
1.2.1.3.2 类图

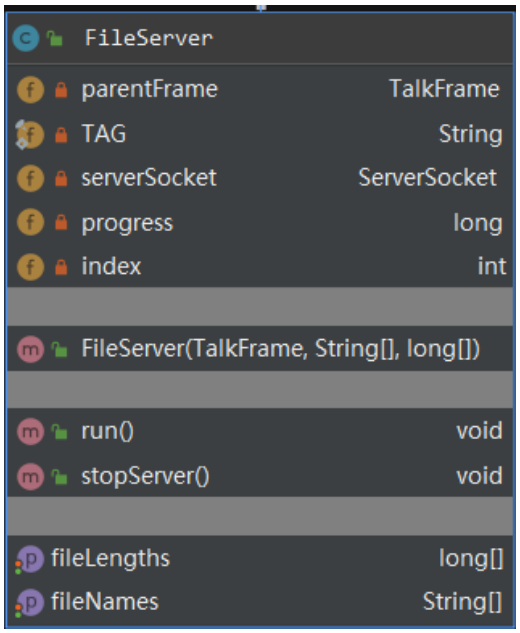
1. FileClient.java





2. FileServer.java

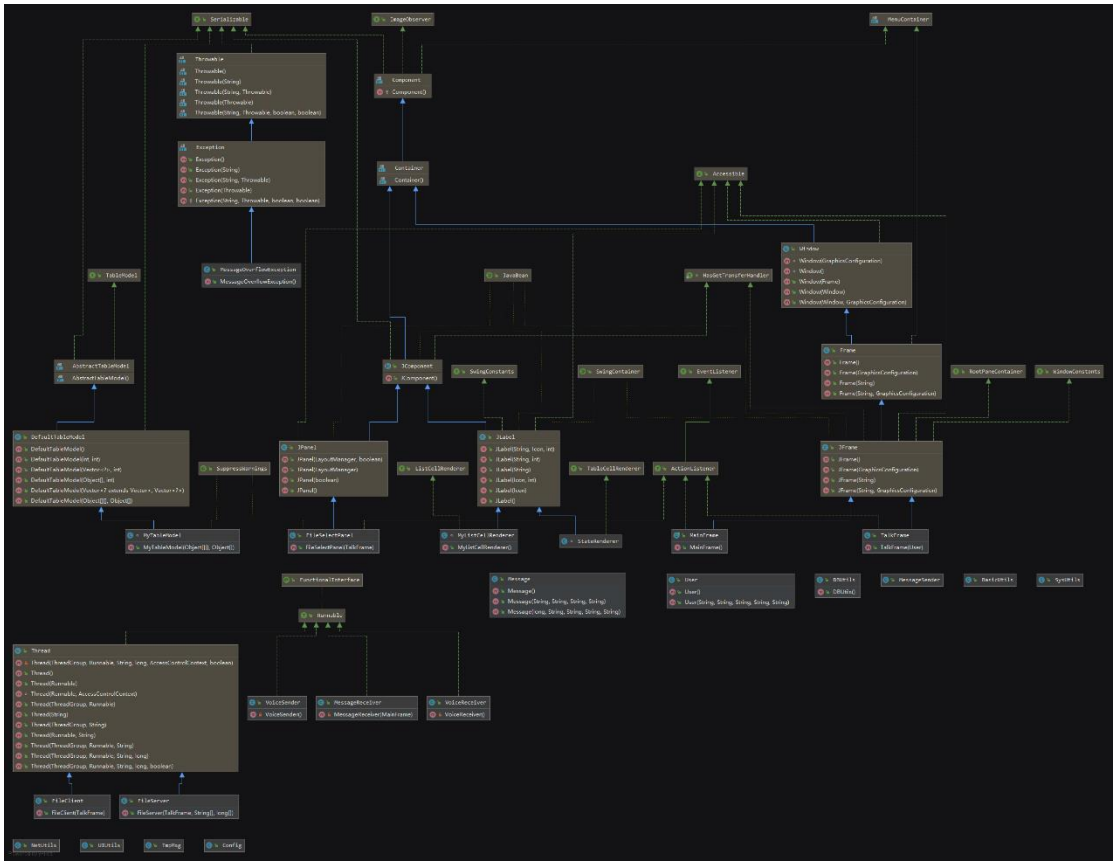

FunctionalInterface
Runnable
Thread
Thread(ThreadGroup, Runnable, String, long, AccessControlContext, boolean)
Thread()
Thread(Runnable)
Thread(Runnable, AccessControlContext)
Thread(ThreadGroup, Runnable)
Thread(String)
Thread(ThreadGroup, String)
Thread(Runnable, String)
Thread(ThreadGroup, Runnable, String)
Thread(ThreadGroup, Runnable, String, long)
Thread(ThreadGroup, Runnable, String, long, boolean)
FileServer
FileServer(TalkFrame, String[], long[])
Powered by yFiles


FileServer
parentFrame TalkFrame
TAG String
serverSocket ServerSocket
progress long
index int
FileServer(TalkFrame, String[], long[])
run() void
stopServer() void
fileLengths long[]
fileNames String[]

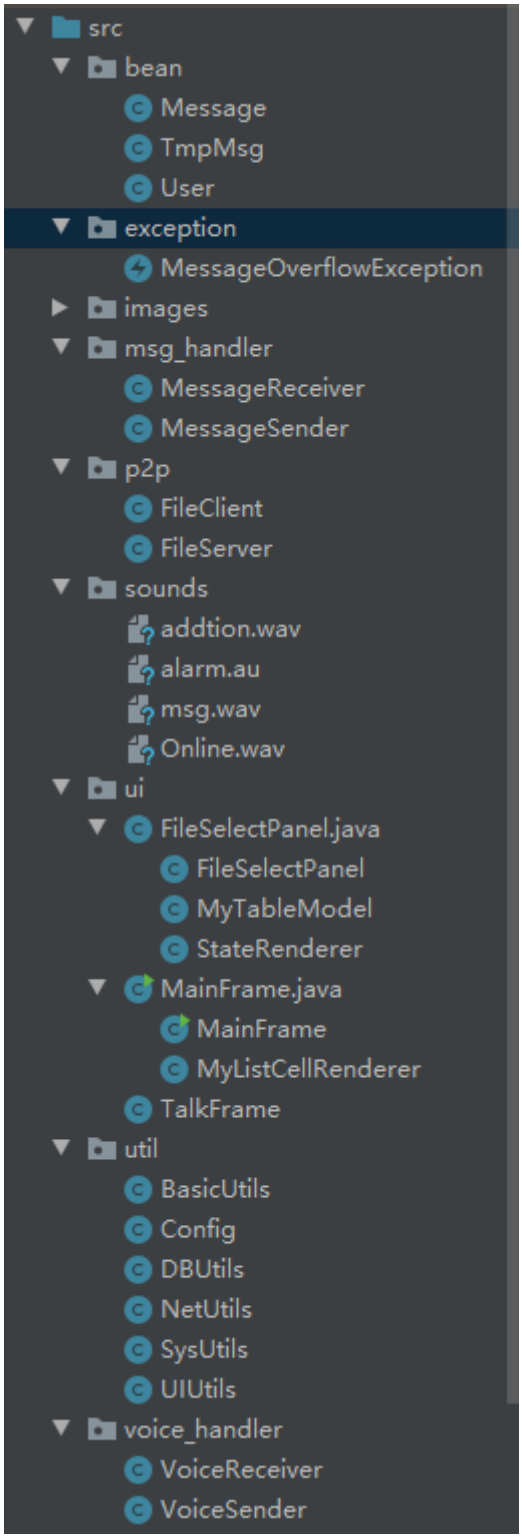
1.2.1.3.3 类设计思路

TCP 长连接进行字节流数据的传输，此部分是单纯的使用 TCP 协议进行三次握手，中间穿插 UDP 的数据提醒，没发现可以使用设计模式改进的部分，也许以后对系统改进之后可以进行文件类别的分类，可以利用工厂模式进行改造

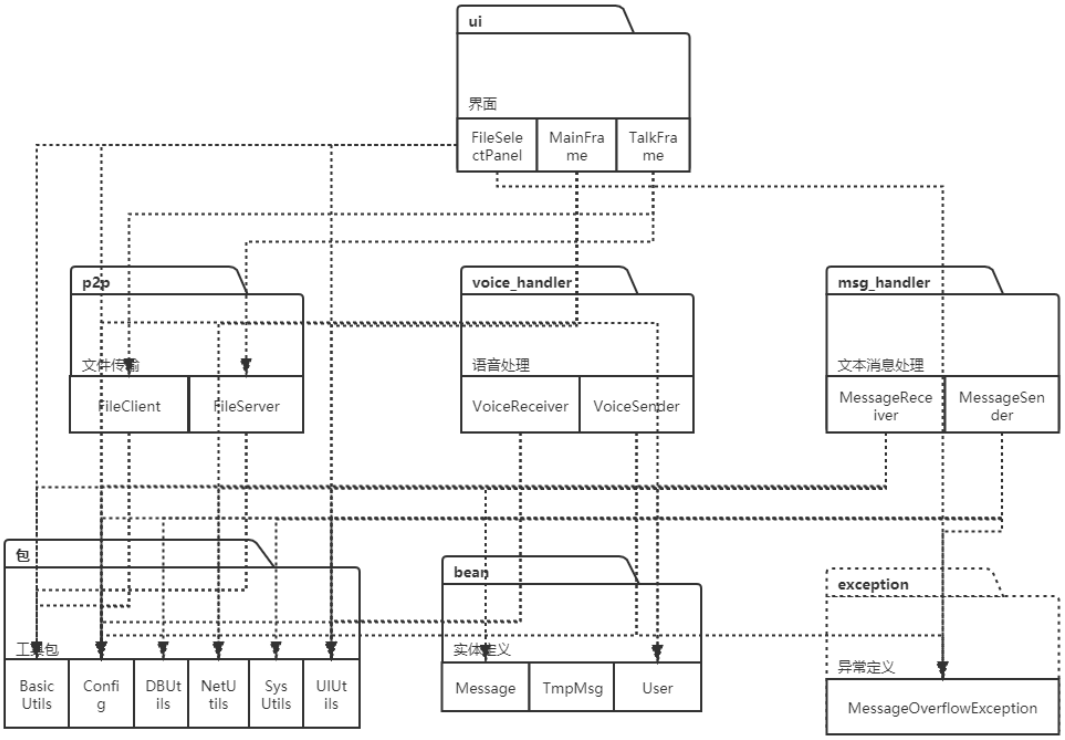
1.2.2 项目类图



1.2.3 项目目录



1. 包的设计
   a) 趋向于共同重用的类属于同一个包
   b) 相互之间没有紧密联系的类不应该在同一个包中
2. 包依赖图



a) 图中未出现环，且符合 DIP 原则

b) 得出包的目录是比较理想的结论

2 使用设计模式拟达到效果

2.1 对测试人员容易测试

2.1.1 容易对每个方法进行单元测试

2.1.2 容易对每个类进行黑盒测试

2.2 对阅读人员容易理解

2.2.1 单个类的代码行数较短

2.2.2 函数加必要的注释

2.3 对开发人员容易拓展

2.3.1 利用工厂模式可以较为方便的拓展

3 拟采取的设计模式

3.1 整体的架构

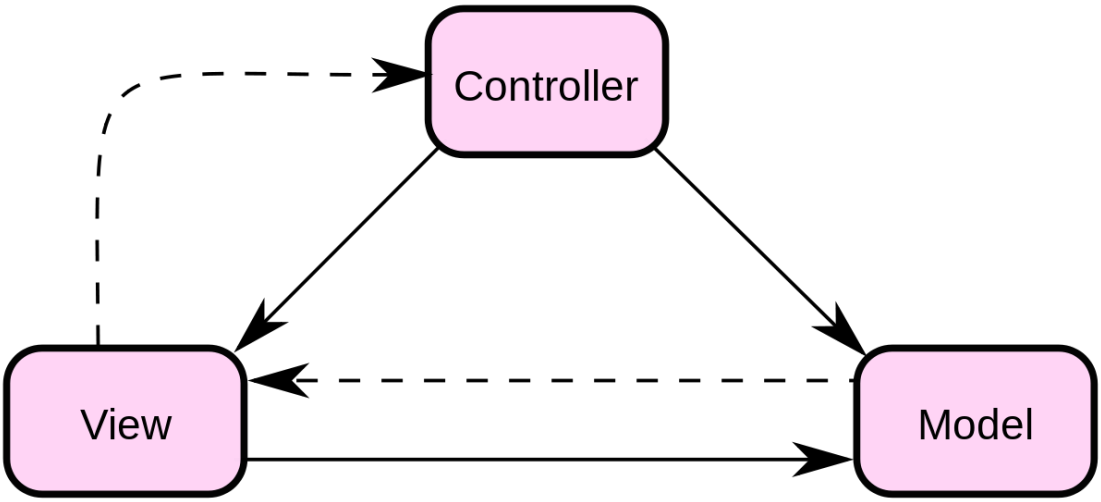
3.1.1 包含数据实体类、前端展示，数据库操作等，所以采用 MVC 模式

3.2 设计模式简介

3.2.1 综上，拟采用的设计模式有 MVC 模式，工厂模式，DCL 双检锁单例模式

3.2.1.1 MVC 模式

1. Model(模型)
   一个存取数据的对象。也可以带有逻辑，在数据变化时更新控制器
2. View(视图)
   模型所包含的数据的可视化
3. Controller(控制器)
   作用域模型和视图上。控制数据流流向模型对象，并在数据变化时更新视图。



3.2.1.2 抽象工厂模式

围绕一个超级工厂创建其他工厂。该超级工厂又称为其他工厂的工厂。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。

在抽象工厂模式中，接口是负责创建一个相关对象的工厂，不需要显式指定它们的类。每个生成的工厂都能按照工厂模式提供对象。

3.2.1.3 DCL 双检锁单例模式

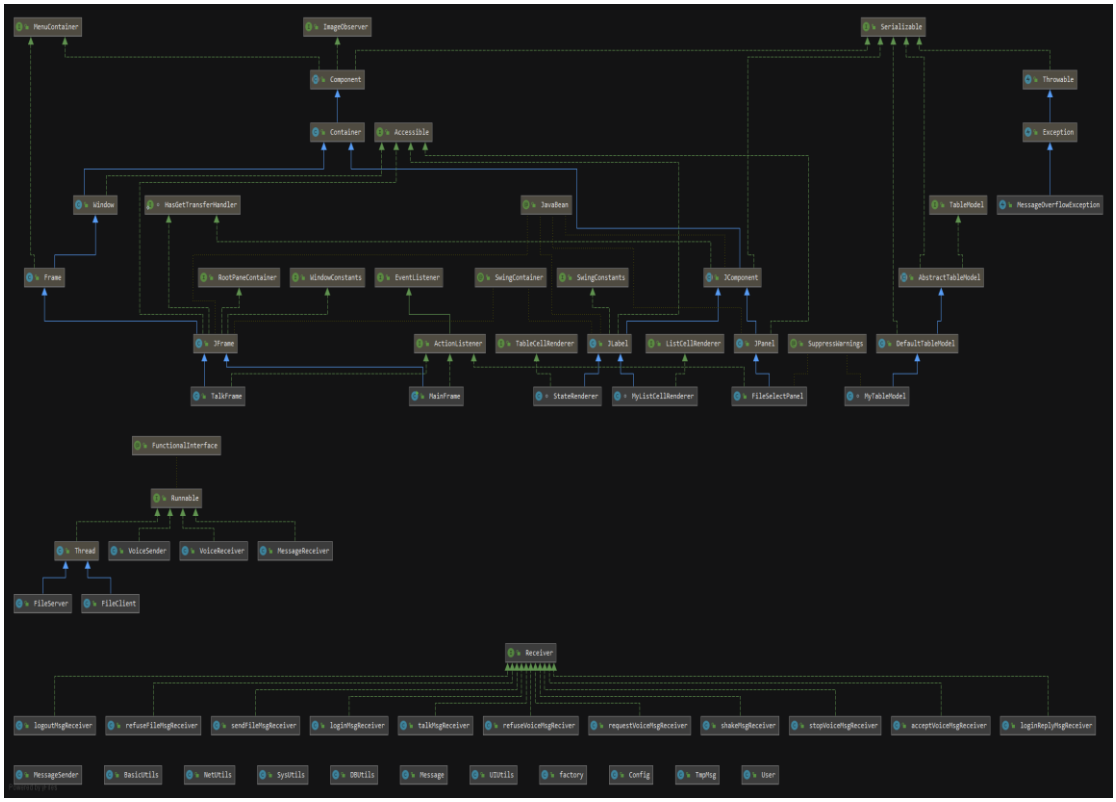
```
public static MessageReceiver getInstance(MainFrame mf){
    if(instance == null){
        synchronized (MessageReceiver.class){
            if(instance == null){
                instance = new MessageReceiver(mf);
            }
        }
    }
    instance.mf = mf;
    return instance;
}
```

```
public static VoiceReceiver getInstance(){
    if(instance == null){
        synchronized (VoiceReceiver.class){
            if(instance == null){
                instance = new VoiceReceiver();
            }
        }
    }
    return instance;
}
```

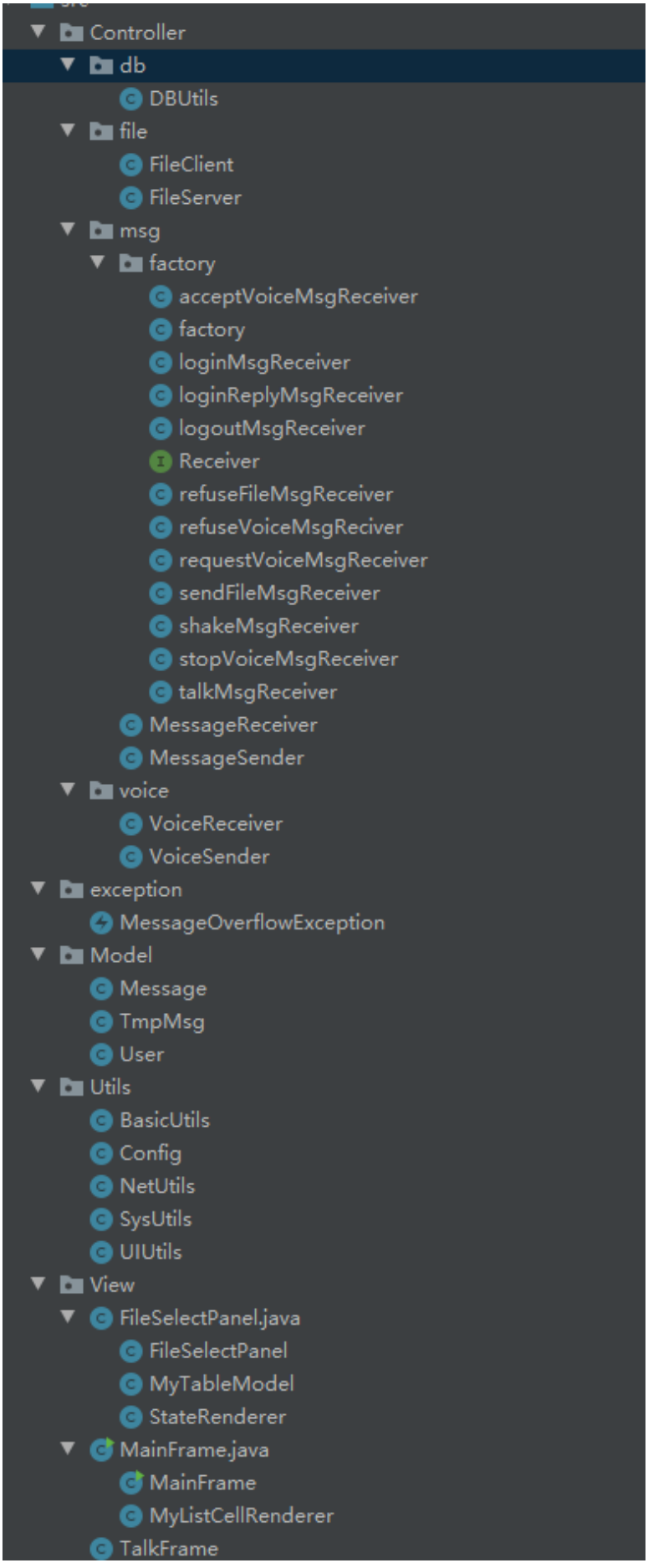
```java
public static VoiceSender getInstance(){
    if(instance == null){
        synchronized (VoiceReceiver.class){
            if(instance == null){
                instance = new VoiceSender();
            }
        }
    }
    return instance;
}
```

4 改进后项目

4.1 项目类图



4.2 项目目录


Controller
db
DBUtils
file
FileClient
FileServer
msg
factory
acceptVoiceMsgReceiver
factory
loginMsgReceiver
loginReplyMsgReceiver
logoutMsgReceiver
Receiver
refuseFileMsgReceiver
refuseVoiceMsgReciver
requestVoiceMsgReceiver
sendFileMsgReceiver
shakeMsgReceiver
stopVoiceMsgReceiver
talkMsgReceiver
MessageReceiver
MessageSender
voice
VoiceReceiver
VoiceSender
exception
MessageOverflowException
Model
Message
TmpMsg
User
Utils
BasicUtils
Config
NetUtils
SysUtils
UIUtils
View
FileSelectPanel.java
FileSelectPanel
MyTableModel
StateRenderer
MainFrame.java
MainFrame
MyListCellRenderer
TalkFrame

5 核心代码展示

5.1 信息发送

```
public static void send(Message msg,String host) throws MessageOverflowException, IOException, DecoderException, SQLException {

    boolean isSensitive= DBUtils.getSensitiveResult(msg);
    if(isSensitive) {
        msg.setData("****");
    }

    byte[] data = msg.toBytes();

    //数据长度限制在512字节内
    if(data.length > 512){
        throw new MessageOverflowException();
    }
    packet = new DatagramPacket(data, data.length, InetAddress.getByName(host),Config.MSG_UDP_PORT);
    socket.send(packet);
}
```

5.2 信息接收

```
socket.receive(packet);
String hostAddress = packet.getAddress().getHostAddress();
Message msg = parseMessage(packet.getData(), packet
        .getLength());
String msgHead = msg.getHead();
String msgData = msg.getData();
String userName = msg.getUserName();
String hostName = msg.getHostName();
User user = new User();
user.setHostAddress(hostAddress);
user.setHostName(hostName);
user.setUserName(userName);
String sendTime = BasicUtils.formatDate(new Date(msg.getSeq()));
String info = userName + " " + sendTime;
boolean isSelf = hostAddress.equals(NetUtils.getLocalHostAddress());
```

5.3 语音发送

```
AudioFormat format =new AudioFormat(AudioFormat.Encoding.PCM_SIGNED, sampleRate: 44100.0f, sampleSizeInBits: 16, channels: 1, frameS
DataLine.Info info = new DataLine.Info(TargetDataLine.class,format);
try {
    targetDataLine = (TargetDataLine) AudioSystem.getLine(info);
    if(!targetDataLine.isOpen()) {
        targetDataLine.open(format, targetDataLine.getBufferSize());
    }
} catch (LineUnavailableException e) {
    e.printStackTrace();
    flag = true;
    System.out.println("说明可能是已经在运行中而被占用，故尝试设置flag为true然后退出");
    return ;
}

int length = 512;
System.out.println(length);
byte[] data = new byte[length];
int readLen=0;
targetDataLine.start();
while (flag) {
    readLen = targetDataLine.read(data, off: 0,data.length);
    try {
        //发送出去
        send(data, readLen);
    }
    catch (Exception ex) {
        ex.printStackTrace();
        break;
    }
}
targetDataLine.stop();
```

5.4 语音接收

```
byte[] data = new byte[512];
packet = new DatagramPacket(data, data.length);

AudioFormat format =new AudioFormat(AudioFormat.Encoding.PCM_SIGNED, sampleRate: 44100.0f, sampleSizeInBits: 16, channels: 1, frameSize: 2, frameRate: 44100.0f, bigEr
//AudioFormat format = new AudioFormat(44100F,16,2,true,true);
DataLine.Info info = new DataLine.Info(SourceDataLine.class,format);
try {
    sourceDataLine = (SourceDataLine) AudioSystem.getLine(info);
    sourceDataLine.open(format, bufSize);
} catch (LineUnavailableException ex) {
    ex.printStackTrace();
    flag = false;
    return;
}
sourceDataLine.start();
while (flag) {
    try{
        socket.receive(packet);
        sourceDataLine.write(packet.getData(), off: 0, packet.getLength());
    } catch (IOException e) {
        e.printStackTrace();
        break;
    }
}
if (flag) {
    sourceDataLine.drain();
}
sourceDataLine.stop();
```

5.5 文件发送

```java
while( ((len=socketIS.read(buffer))!=-1){
    String msg = new String(buffer, offset: 0,len);
    //对方发送来的信息是个整数，表示将要接收的文件的编号；
    index = Integer.parseInt(msg);
    System.out.println("对方发来的序号是："+index);
    //当传送完最后一个文件会再发送一个整数信息，其值等于文件个数，表明已全部接收完毕，则关闭服务器
    if(index==(fileNames.length)){
        System.out.println("即将退出循环");
        break;
    }
    System.out.println(TAG+"=========================================================");
    System.out.println(TAG+"开始传送第【"+index+"】号文件："+fileNames[index]+"，请等待...");
    long startTime = System.currentTimeMillis();
    this.parentFrame.updateSendTableState(index, Config.STATE_SENDING);
    File fileToSend = new File(fileNames[index]);
    BufferedInputStream bis = new BufferedInputStream(new FileInputStream(fileToSend));
    this.progress = 0;
    long fileLength = fileLengths[index];
    while((len=bis.read(buffer))!=-1){
        socketOS.write(buffer, off: 0, len);
        //进度信息
        progress += len;
        String lengthShow = BasicUtils.convertByte(fileLength);
        String info = "("+(index+1)+"/"+fileNames.length+")  "+BasicUtils.convertByte(progress)+"/"+lengthShow+"  "+progress*100/fileLength+"%";
        int pValue = (int)(progress*100/fileLength);
        parentFrame.updateSendProgress(pValue,info, toolTip: "正在发送："+fileNames[index]);
    }
    socketOS.flush();
    bis.close();
    System.out.println(TAG+"第【"+index+"】号文件传送完毕～");
    long costTime = (System.currentTimeMillis()-startTime);
    totalCostTime += costTime;
    totalFileCount++;
    totalFileLength += fileLengths[index];
    parentFrame.updateSendTableState(index, Config.STATE_COMPLETED);
    parentFrame.appendSysMsg("文件["+fileNames[index]+"]发送完毕，文件大小["+BasicUtils.convertByte(fileLengths[index])+"]，耗时["
            +BasicUtils.convertTime( second: costTime/1000)+"] "+BasicUtils.formatDate(new Date()));
```

5.6 文件接收

```java
long totalFileLength = 0; //总文件大小
while(index<fileNames.length){
    int len = 0;
    long progress = 0;
    long fileLength = fileLengths[index];
    long startTime = System.currentTimeMillis();
    socketOS.write(String.valueOf(index).getBytes()); //先发送一条消息，通知服务器下一条要接收的文件编号
    socketOS.flush();
    this.talkFrame.updateReceiveTableState(index, Config.STATE_ACCEPTING);
    String filePath = this.savePath+"\\"+fileNames[index];
    BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(new File(filePath)));
    while(progress<fileLengths[index]  && ((len=socketIS.read(buffer))!=-1){
        bos.write(buffer, off: 0, len);
        //进度信息
        progress += len;
        String lengthShow = BasicUtils.convertByte(fileLength);
        String info = "("+(index+1)+"/"+fileNames.length+")  "+BasicUtils.convertByte(progress)+"/"+lengthShow+"  "+progress*100/fileLength+"%";
        int pValue = (int)(progress*100/fileLength);
        talkFrame.updateReceiveProgress(pValue,info, toolTip: "正在接收："+fileNames[index]);
    }
    bos.flush();
    bos.close();
    System.out.println(TAG+"第【"+index+"】号文件接收完毕～");
    long costTime = (System.currentTimeMillis()-startTime);
    totalCostTime += costTime;
    totalFileCount++;
    totalFileLength += fileLengths[index];
    this.talkFrame.updateReceiveTableState(index, Config.STATE_COMPLETED);
    talkFrame.appendSysMsg("文件["+fileNames[index]+"]接收完毕，文件大小["+BasicUtils.convertByte(fileLengths[index])+"]，耗时["
            +BasicUtils.convertTime( second: costTime/1000)+"] "+BasicUtils.formatDate(new Date()));
    index++;
}
```