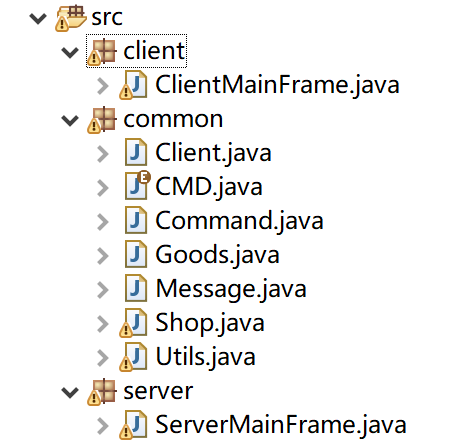
1. 系统结构



系统分为3个Package，分别为client、common和server。其中client下ClientMainFrame即客户端程序源代码，server下ServerMainFrame即服务器端源代码。common下面有Client、CMD、command、Goods、Message、Shop、Utils。Client类为客户，将用户名、用户IP地址、用户端口、客户拥有的店铺进行封装。CMD为客户端与服务端交互的协议。Command类将用户在TextArea中输入的命令和在combobox中选取的接受客户/商店名进行封装。Goods类将goodsID、goodsName、goodsPrice进行封装。Message类为其中最重要的类。属性Carrier表示载体，不同的命令对应不同的载体，可能是Vector<Goods>、Vextor<Client>，也有可能是Command。CMD表示交互用的协议。Client一般是请求客户。Message类似于应用层报文。Shop类下属性有店铺的商品存储在Vector<Goods>中，正在店铺中逛的顾客存储在Vector<Client>中，店铺的拥有者，和店铺名，店铺名和ClientIP.getHostName().toString()相同。Utils类下包含了通过UDP发送和接受Message的函数。

1. 应用程序协议



登录：客户端发送USERLOGIN,如用户名存在，服务端应答USERALREADYEXISTS,否则应答LOGINSUCCESS。

客户端发送请求SENDSHOPLIST，服务端应答SENDSHOPLIST并捎带回shopList。

客户端发送请求SENDMESSAGE并捎带命令行信息，服务端收到SENDMESSAGE通过解析命令行，根据命令行内容，或应答SENDSHOP并捎带商店信息，或应答SENDGOODS并捎带商店商品信息，或应答SENDCUSTOMERS返回正在逛的顾客信息，或应答LEAVESHOP并更新shopList，或应答ENTERSHOP,或应答SENDGOODSSOLD返回已出售商品信息。客户端收到相应协议内容运行协议相应的程序。同理，服务器端发送SENDCOMMAND并稍带命令行信息，客户端收到SENDCOMMAND对命令行进行解析。根据解析内容向执行代码，例如发送CLOSE，关闭商城。

SERVERSTOP：表示服务器停止服务。

SERVERSTOP：表示客户端退出。

1. 关键代码说明

1.Client类中：

@Override

**public** **boolean** equals(Object obj) {

**if**(obj==**null**)

**return** **false**;

//如果用户的姓名一样，则它们相同

**if**(obj **instanceof** Client){

**return** clientName.equals(((Client)obj).clientName);

}

**return** **false**;

}

如果ClientName相同则两Client相同，便于从Combobox中通过ClientName查找Client。

Shop和Goods中对equals方法的重写同理。

2. //绑定服务器端口，然后接受来自客户端的信息

Message msg = Utils.*receiveMessage*(*receiveSocket*);

3.将Carrier中携带的商品信息强制转换，存入goodsAvailable。

Vector <Goods> goodsAvailable = (Vector <Goods>)msg.getCarrier();

4. 从Combobox中通过ClientName查找Client。

Client receiveClient1 = *userList*.elementAt(*userList*.indexOf(**new** Client(receiveClientName,**null**,0)));

**5.服务器端为正在商店逛的用户和商店拥有者更新商品信息**

**for** (Client c : requestShop.getCustomers()) {

Message msg8 = **new** Message(requestShop.getGoods(), CMD.***CMD\_SENDGOODS***, **null**);

Utils.*sendMessage*(msg8, c.getClientIp(), c.getClientPort());

}

Message msg9 = **new** Message(requestShop.getGoods(), CMD.***CMD\_SENDGOODS***, **null**);

Utils.*sendMessage*(msg9, requestClient.getClientIp(), requestClient.getClientPort());

6. 增加商品

String[] tmp1 = text.split(" ");

String goodsID = tmp1[1];

String goodsName = tmp1[2];

**double** goodsPrice = Double.*valueOf*(tmp1[3]);

requestShop.addGoods(**new** Goods(goodsID, goodsName, goodsPrice));

7. //由于本地测试时，客户端的端口号要不一致

**private** **static** **int** *clientPort* = **new** Random().nextInt(10000)+1024;

8. //将消息格式内容，命令，用户的地址信息综合成msg,并发送给服务器。

Message msg1 = **new** Message(myShop, CMD.***CMD\_SENDSHOP***, **null**);

Utils.*sendMessage*(msg1, InetAddress.*getByName*(Utils.*serverIP*), Utils.*serverPort*);

9. **服务器向用户列表发送协议。**

**for**(Client c : *userList*){

Message tm = **new** Message(**null**, CMD.***CMD\_SERVERSTOP***, **null**);

**try** {

Utils.*sendMessage*(tm,c.getClientIp(),c.getClientPort());

} **catch** (Exception e1) {

e1.printStackTrace();

}

}