**The university of Melbourne**

***SCHOOL OF COMPUTING AND INFORMATION SYSTEMS***

# *COMP90015 Distributed Systems*
# *Assignment 2 Report*
# *The Implementation of the Distributed whiteboard*

**Changwen Li 1360219**

**Xunhai Wang 941106**

## *Executive Summary*

In this project, we have implemented all required features (both basic and advanced features). We use the hybrid architecture, request-based socket, and thread pool to implement these features.

The hybrid architecture in this project is the combination of client-server architecture and peer-to-peer architecture. The reason we do not use pure client-server architecture is that our server is weak in computational power. There are only two cores on the server while most of the current PCs have 4 or 8 cores. With the technology of hyper-threading, most PCs have 8 or 16 parallel processing units. The computational power on PCs is usually much higher than on our server. Therefore, it is natural to exploit peers' stronger computational power. In our project, the server plays the role of recording user list and manager info, while peers will pass their drawing/chat message directly to other peers.

We choose request-based sockets instead of connection-based sockets to implement our whiteboard. The request-based socket means that the sockets are created every time there is a request. For example, the user will build a socket to send a request to the server, and the socket is closed once the request is finished. The socket connection will not last long. We assume that the frequency of message passing including user list updates, whiteboard synchronization, and other operations is low. Users are idle most of the time. Under this presumption, it will be unwise to hold the socket connection to waste system resources waiting for information.

We use thread and thread pool. A Thread pool has fewer overheads than thread. In the thread pool, each thread in the pool will only be created once and they will not be destroyed even after they have finished their assigned tasks. On the contrary, the thread will be destroyed once it has finished its work. If we use thread, then we may need to frequently create and destroy threads, which can be time-consuming. In our implementation, we use a thread pool to send and respond to chat messages and drawing messages. The most time-consuming jobs are executed by the thread pool. We also use thread to deal with a light workload since the thread is easier to implement.

## *3.Table of Contents*

## *4.Architecture*

We have used the hybrid of server-client and peer-to-peer architectures. The reason we adopt this approach is that it can achieve higher performance. A detailed explanation has been introduced in the executive summary. The server is responsible for recording the user list and manager info. When the user list is empty, the first client connected to the server will become the manager. The user's username, IP address, and port number will be used to identify a user's identity uniquely. Every time an alternation of the user list (new users join in or old users leave) will trigger, the server sends the user list to every client. Peers then update their local user lists. Peers are responsible for sending and receiving messages of chat/drawing. Every time the peer sends a chat/drawing, it will broadcast this message to everyone.
There are two threads and a thread pool with eight threads. The two threads are IOThread and WorkThread. IOThread gets the incoming sockets and store them to a thread-safe queue. WorkThread then reads and dequeues from the socket queue. It will then respond to these sockets. The light-workload task will be done by WorkThread, such as receiving manager information and updating userList. The thread pool executes the heavy workload functions such as synchronizing drawboard and chat room.

## *5.Communication Protocol*

There are many message-passing protocols used in this project. The protocols of user list update and drawing synchronization have been discussed in the architecture section and executive summary section. In this section, we discuss the protocols of manager management, such as the drawboard record file opening protocol and access approve/deny protocol.
The manager has the right to load a drawboard record file. After loading the record file, every user needs to see the content of the record file on their board. The protocol to implement this is setting up a queue to store the messages from the record file. Then the manager will broadcast these messages to all peers. Please notice that this message queue will record all messages regarding drawing. When a new user joins the system, the manager will pass the information to this peer.
The finite state machine graph below demonstrates the state change of the new user who is not the manager. The latest user sends the request to the server. The server passes the request to the manager, who will decide if he wants to approve/deny the access. If he denies the request, the manager will send a message directly to this user. If he approves the request, the manager will send a message to the server, indicating a change in the user list. Then, he will send history records to the user.

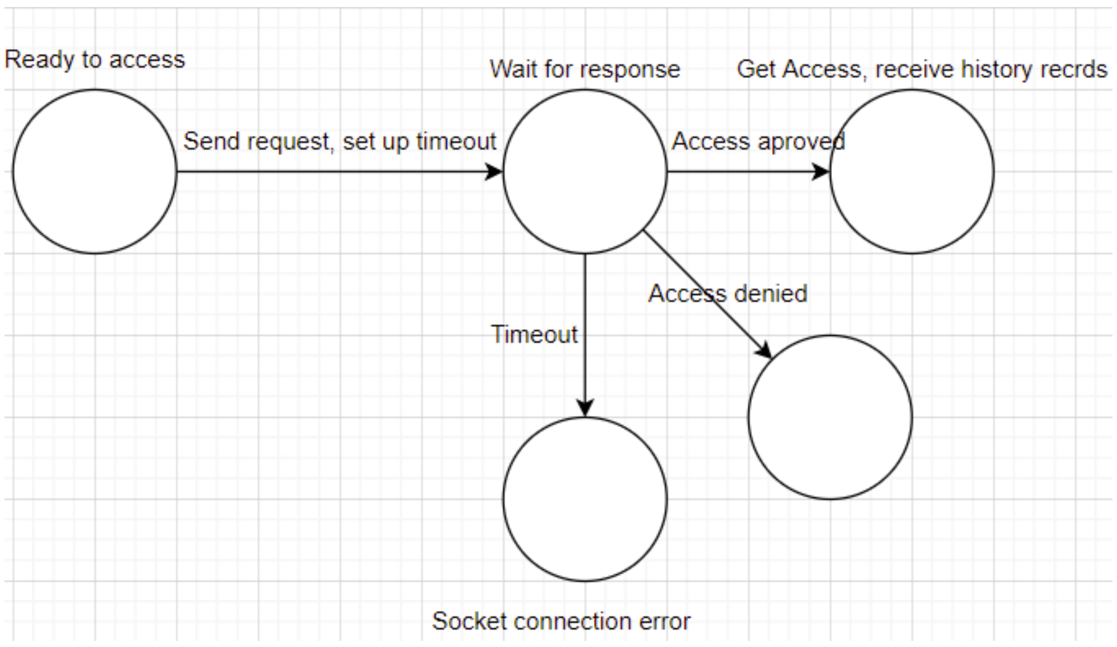


Figure 1: finite state machine of a new user during access request

## *6.Message Formats*

This article frequently mentions the message passing between server and client, the variable information used in message passing such as massage, MsgName, etc. In fact, in the function implementation, all messages are passed in the form of JSON strings. , the content of the transmitted message uniformly encapsulates the following parameters. The data is obtained when sending and serialized into a JSON string. After receiving the message at the receiving end, it is deserialized into a JSONObject object, and the data is obtained from the JSON object. When implementing different messages, the processing methods of parsing data when sending and receiving are consistent.

```
//message information
jsonObject.put("MsgName", "PermissionRequest");
//username
jsonObject.put("username", username);
//socket port
jsonObject.put("userServerPort", Integer.valueOf(userServerPort));
//ip
jsonObject.put("userIP", ip.getHostAddress());
```

# 7 Implementation Details

## 7.1 Implementation of the server-client connection

We first start the server and initialize the variables.

```java
private static String username = "admin"; //username
private static int port = 3200; //ServerSocket port
private static InetAddress serverIP;  //ip
//Client connection Socket queue
LinkedBlockingDeque<Socket> socketQueue = new LinkedBlockingDeque<>();
//Client user queue
LinkedBlockingDeque<ID> userList = new LinkedBlockingDeque<>();
```

After the initialization of the variables, the main method starts to create IOThread Object, which extends the Thread class. In this IOThread, create the ServerSocket Object and call the accept method, waiting for the join request from the client to establish the connection. After establishing the connection between the server and the client, set a timeout for this connection and add this connection to the initialized LinkedBlockingDeque socketQueue. In this thread, execute these operations above with a while loop until any interruption of the thread. In the execution process, if any IOException is caught, end the loop and call the close method and close the serverSocket connection.

while the socketQueue is not empty, pop the first element from the socketQueue. Put the Socket received and the initialized userlist into the ServerSocketReceiver object to create a new ServerSocketReceiver object. After that, call the response method in the ServerSocketReceiver to handle the information of the client connected to the server. Finally, close the socket. Keep executing the above operations until the socketQueue is empty.

Inside the ServerSocketReceiver object, when the object is created through the parameterized constructor, the socket object and userlist collection are obtained through parameters, and assigned to their own global variables, the input stream is obtained through the socket, and the link of the link is read through the input stream. Client information, and assign values to the following two fields:

```java
private String MsgName;
private JSONObject command;
```

When calling the response method, determine whether MsgName = SendJoinRequest. Create a updateUserListSender object to handle requests if true.

When the updateUserListSender object is created, the following data will be obtained

```java
private Socket socket; //client socket
private JSONObject command; //link info
private LinkedBlockingDeque<ID> userList; //client user info list
```

At the same time, the client's InetAddress will be obtained through the socket, and the socket connection will be closed, and then the username and userServerPort attribute values will be obtained through the command parameter, and an ID object will be created to save the detailed information of the Socket Client link. Add the created ID object to the userList, and finally call the sendUpdates method of the updateUserListSender to notify all the added client-side userList data changes. In the sendUpdates method, the IDQueueToString method in the tool class Translator_IDQ_JSStr is used, and the processed The queue userList is converted to JSON string type, after getting the converted string, loop the userList queue, get each ID object in the queue, get the details of the client link through the ID object, create a socket link, and get the output stream , write the converted string to the client, and close the connection.

## *7.2 Implementation of the simutanous drawboard*

When starting the drawing board, in addition to opening the content of the drawing board itself, it also binds event listeners for painting, and establishes two threads at the same time. The first IOThread thread is used to start the server side, and calls the accept method to block and wait for the client side connection. , and add the client information to the LinkedBlockingDeque<Socket> socketQueue; save it in the queue to record all the clients that establish the connection (the specific implementation is consistent with the logic in the first paragraph).

After the brush triggers the event, the painting content will be obtained, and the painting content will be parsed as a string. When the event is triggered, a Socket link will be created immediately, and the content parsed by the brush will be sent to all the clients in the link (the message sending specific The logic is exactly the same as the first paragraph, but the message body sent is changed to the string parsed from the painting content).

The second WorkThread thread is used to monitor the messages sent by other artboards. When the WorkThread is created, the following properties need to be obtained:

```java
private ExecutorService pool;
private LinkedBlockingDeque<Socket> socketQueue;
private LinkedBlockingDeque<ID> userList; // peer userList!
private DrawBoard drawBoard;
```

The difference from the first paragraph is that after the WorkThread listens to the message, it needs to re-convert the message into the data required by the drawing board and write it back to the drawing board. Therefore, the drawing board object is first obtained in the WorkThread to operate the drawing board, because the drawing board message The receiving frequency is very high. In order to improve efficiency, a task thread pool of ExecutorService is obtained to execute the task of writing back the drawing board data.

The PeerSocketReceiver object is created in the WorkThread to obtain the received message from the socket's input stream and serialize the JSON, parse the required data (consistent with the logic of the first paragraph), and then call the response method to pass the received drawing board message. Create a SyncDraw object, get the drawing board message in the SyncDraw object, and execute the drawBoard.receiveText() method, refresh the brush data to the drawing board, and finally put the SyncDraw object into the pool task thread pool, pass pool.submit(new SyncDraw() ) method submits and executes the contents of the SyncDraw object.

The above implementation logic explains that the implementation of synchronous drawing on the drawing board is in the form of p2p. Each drawing board is a server and a client at the same time, so it is realized that any drawing board is directly linked and does not need to be independent. The server side transfers data, which improves communication efficiency. At the same time, the second advantage is that this implementation does not always maintain a state of constant linking but creates a new link for a session every time a request is triggered, and the session is completed. Close the link immediately after the connection, although the frequent establishment and closing of the link will have a certain performance loss, the security and reliability of the communication link are guaranteed, and there is no need to worry about the possibility of message loss due to the failure of reconnection after the link is disconnected.

## *7.2 Implementation of the Online Userlist*

The online user list technology is completely based on the server to client technology described in the first paragraph. After the server service is started, an independent IOThread thread will be opened, a serverSocket object will be created in this thread, and the accept method of the serverSocket object will be called to block and wait for new the client accesses and stores the client information in the blocking queue socketQueue, and finally closes the socket link. The above logic is the same as the description in the first paragraph, but in the subsequent processing of the client client link information in the socketQueue, the logic is added.

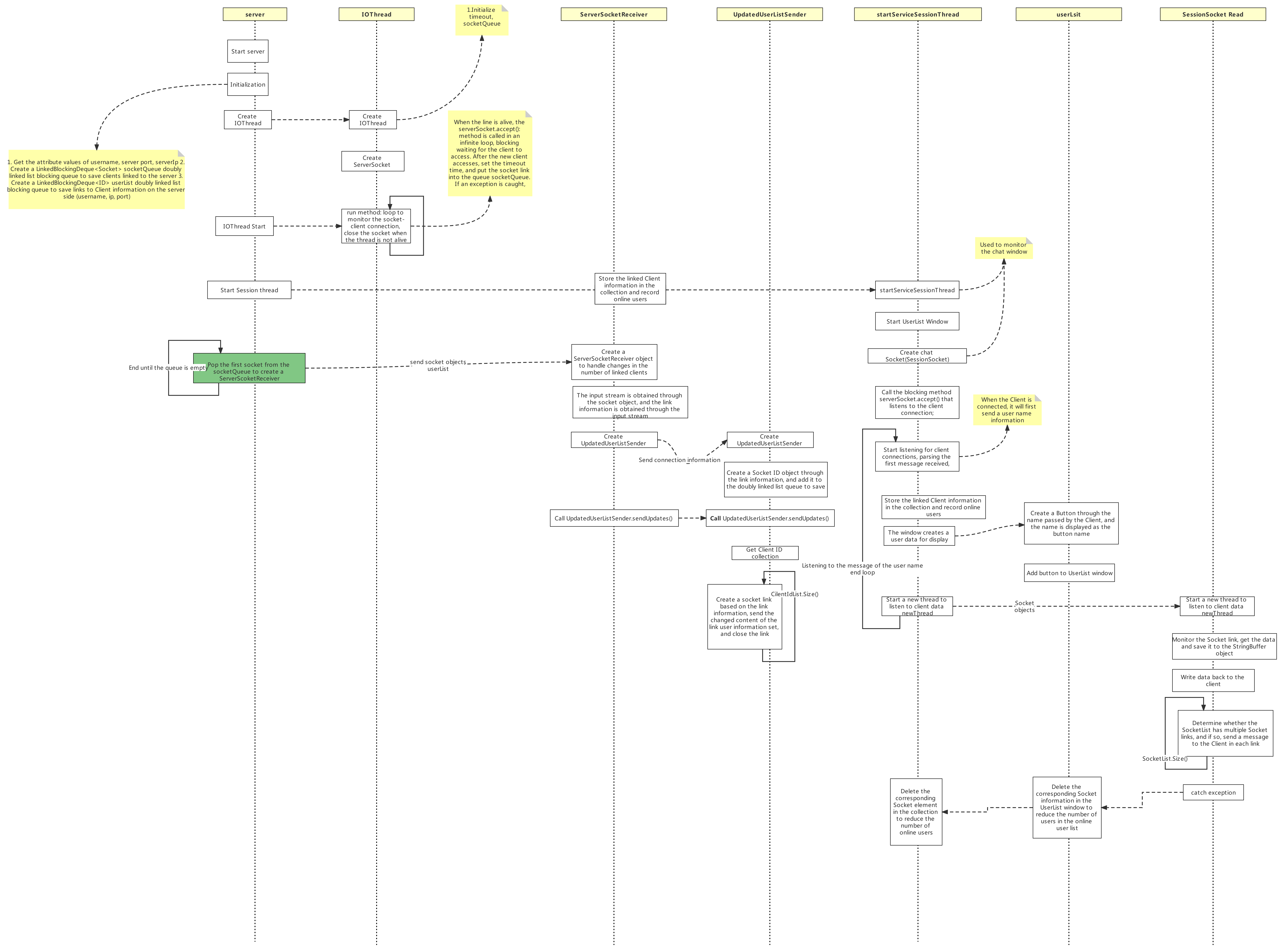
First, the processing logic for monitoring the establishment and disconnection of the client-side connection is added to the original ServerSocketReceiver object. When the ServerSocketReceiver object is created, the received message data is parsed, and the response() method of the object is called at the same time. In the response() method, different business logics are processed according to the received different MsgName identifiers.

When the received MsgName = SendJoinInRequest, the business logic of the new client joining the link is processed. At this time, it is judged whether the user list is empty to confirm whether the client currently joining the link is the first one, and if so, set the first client to Administrator, and save the client-side information as the administrator. If not, create a PermissionRequestSender object. When the object is created, interpret the received client-side information, and then call the send() method to notify the manager that there is a new client-side Access link, ask the manager whether to allow new clients to access, this is after the manager receives the query message, create a query window, the manager can click to choose to allow or kick out the client link, when the manager clicks, through When the click event is triggered, a socket link is created to send the message ID that allows or rejects the link. After the server side receives the message ID, it processes the permission or denial of the client-side link through the ID field.

When processing logic in ServerSocketReceiver, no matter what the received MsgName value is, an UpdatedUserListSender object will be created to process, add or delete client-side links. The link information is added to the online user list, and the change in the number of online users on the client side in all links is notified. When the UpdatedUserListSender object is created, the content of MsgName will be obtained to determine the selection of different business processing logic.

If the MsgName received by the UpdatedUserListSender object is to allow users to join, the user information will be processed, encapsulated into an ID object, and the client-side link information will be added to the userList, and sendUpdates() will be called to send a message to all online clients. changes in the number of users.
If the received data is denied access, the client-side message will be obtained and deleted from the userList collection, and no other operations will be performed.

The function implementation of the online user list does not reflect much on the client side, but when the client side connects for the first time, it will send a jion message to the server side. After the server side parses the message, it notifies the manager, and the manager chooses whether to allow the client to join the link. After receiving the selection message, the process of allowing or denying the user connection is processed, but for the performance of the client side, there are only two responses. If the notification of the connection allowed by the server side is received, after the link is established successfully, it will immediately receive the server side. Online user list data. At this time, the client needs to parse the received online client data and write the data to the userList window. If the manager rejects the client link, the client link will be disconnected directly and will not be added to the online user list. Imperceptible to clients who are denied connection establishment.


server
IOThread
1.Initialize timeout, socketQueue
ServerSocketReceiver
UpdatedUserListSender
startServiceSessionThread
userLsit
SessionSocket Read
Start server
Initialization
Create IOThread
Create IOThread
When the line is alive, the serverSocket.accept(): method is called in an infinite loop, blocking waiting for the client to access. After the new client accesses, set the timeout time, and put the socket link into the queue socketQueue. If an exception is caught,
1. Get the attribute values of username, server port, serverIp 2. Create a LinkedBlockingDeque<Socket> socketQueue doubly linked list blocking queue to save clients linked to the server 3. Create a LinkedBlockingDeque<ID> userList doubly linked list blocking queue to save links to Client information on the server side (username, ip, port)
Create ServerSocket
IOThread Start
run method: loop to monitor the socket-client connection, close the socket when the thread is not alive
Used to monitor the chat window
Start Session thread
Store the linked Client information in the collection and record online users
startServiceSessionThread
Start UserList Window
Create chat Socket(SessionSocket)
End until the queue is empty
Pop the first socket from the socketQueue to create a ServerScoketReceiver
send socket objects userList
Create a ServerSocketReceiver object to handle changes in the number of linked clients
Call the blocking method serverSocket.accept() that listens to the client connection;
When the Client is connected, it will first send a user name information
The input stream is obtained through the socket object, and the link information is obtained through the input stream
Create UpdatedUserListSender
Send connection information
Create UpdatedUserListSender
Start listening for client connections, parsing the first message received,
Create a Socket ID object through the link information, and add it to the doubly linked list queue to save
Store the linked Client information in the collection and record online users
Create a Button through the name passed by the Client, and the name is displayed as the button name
Call UpdatedUserListSender.sendUpdates()
Call UpdatedUserListSender.sendUpdates()
The window creates a user data for display
Get Client ID collection
Add button to UserList window
Listening to the message of the user name end loop
CilentIdList.Size()
Create a socket link based on the link information, send the changed content of the link user information set, and close the link
Start a new thread to listen to client data newThread
Socket objects
Start a new thread to listen to client data newThread
Monitor the Socket link, get the data and save it to the StringBuffer object
Write data back to the client
Determine whether the SocketList has multiple Socket links, and if so, send a message to the Client in each link
SocketList.Size()
Delete the corresponding Socket element in the collection to reduce the number of online users
Delete the corresponding Socket information in the UserList window to reduce the number of users in the online user list
catch exception