計算機網路

Socket聊天室

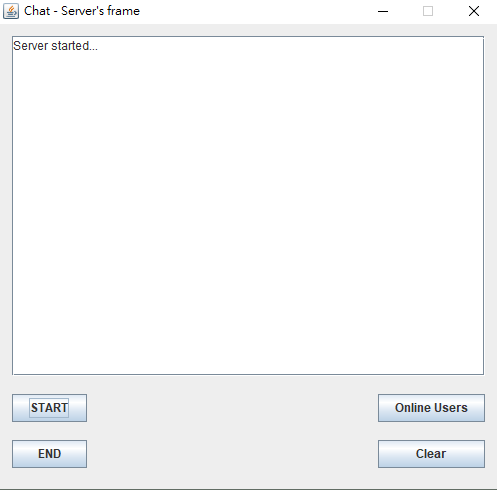
資工三

B0529048

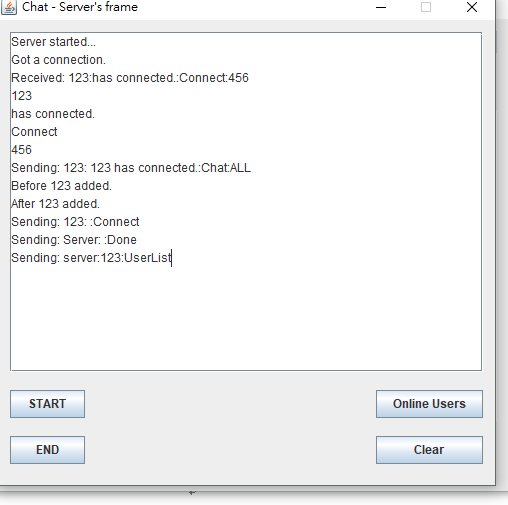
陳列德

1. 操作說明: 說明程式的操作方式

server



1. Start:需先開啟server 並點start啟用server和port



1. 中間textfield :

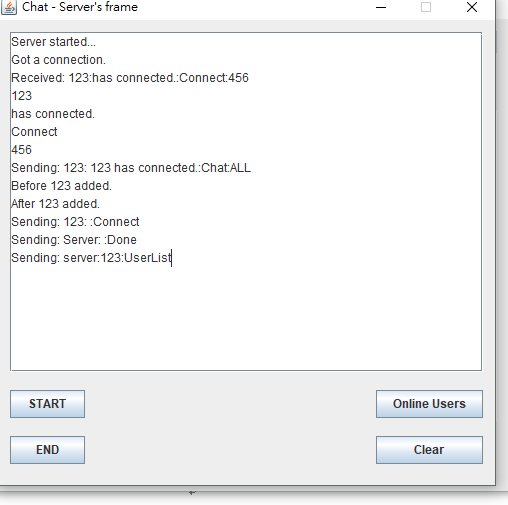
來顯示經由port傳來的訊息

並透過訊息用”:”來切割並定義其對應的操作

由scrollpane來避免訊息超出範圍

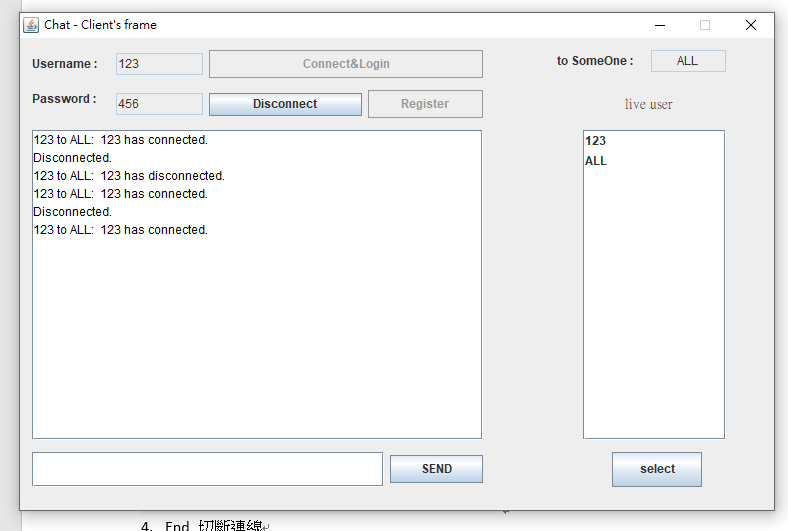
1. Online Users

顯示有誰在線



1. End 切斷連線
2. Clear 清空textfield

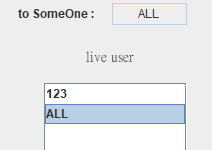
chat



1. 打入username 和 password 按connect來連線
2. 如果成功會讓按鈕不可使用並顯示在現有誰
3. 文字打在最下方裡並按send寄出

4.右方可選要寄給誰再按select確定，可選擇單一人或是在線全部

選擇要送出得人會顯示在 to Someone裡



可多開使用者端來模擬多人在線及指定人傳訊

1. 程式說明: 說明程式的架構與功能, 若有使用特別的技巧也請說明

此程式是用java的netbeans 來撰寫，用裡面的 JFrame 和design來直接拉元件和排版，將目的位置設為localhost和先固定好port 來進行socket連線，並且一次傳遞訊息再由”:”分割，來判斷傳遞形式、內容、以及目的

如: user:message:function:password or destination

並交由server來判斷並做出對應指令再回傳，如錯誤訊息或是成功等等，帳號密碼是由server來儲存，並由回傳訊息來給client判斷是否開方權限

1. 程式: 請使用較小字體與單行行距將程式列印出來

使用者端:

1. package chat\_client;
2. import java.net.\*;
3. import java.io.\*;
4. import java.util.\*;
5. import javax.swing.DefaultListModel;
6. public class client\_frame extends javax.swing.JFrame
7. {
8. String username, address = "localhost",password;
9. ArrayList<String> users = new ArrayList();
10. int port = 2222;
11. Boolean isConnected = false;

14. Socket sock;
15. BufferedReader reader;//br
16. PrintWriter writer;
18. //--------------------------//
20. public void ListenThread()
21. {
22. Thread IncomingReader = new Thread(new IncomingReader());
23. IncomingReader.start();
24. }
26. //--------------------------//
28. public void userAdd(String data)
29. {
30. users.add(data);
31. for(int i = 0; i < users.size(); i++) { //check
32. System.out.println(i+":"+users.get(i));
33. }
34. }
36. //--------------------------//
38. public void userRemove(String data)
39. {
40. ta\_chat.append(data + " is now offline.\n"); //add word at last local
41. }
43. //--------------------------//
45. public void writeUsers()
46. {
47. String[] tempList = new String[(users.size())];
48. users.toArray(tempList);
49. for (String token:tempList)
50. {
51. //users.append(token + "\n");
52. }
53. }
55. //--------------------------//
57. public void sendDisconnect()
58. {
59. String bye = (username + ": :Disconnect");
60. try
61. {
62. writer.println(bye);
63. writer.flush(); //clean and send all message
64. } catch (Exception e)
65. {
66. ta\_chat.append("Could not send Disconnect message.\n");
67. }
68. }
69. //--------------------------//
71. public void Disconnect()
72. {
73. try
74. {
75. ta\_chat.append("Disconnected.\n");
76. sock.close();
77. } catch(Exception ex) {
78. ta\_chat.append("Failed to disconnect. \n");
79. }
80. isConnected = false;
81. tf\_username.setEditable(true);
82. tf\_password.setEditable(true);
83. b\_connect.setEnabled(true);
84. b\_register.setEnabled(true);
85. }
87. public client\_frame()
88. {
89. initComponents();
90. DefaultListModel model = new DefaultListModel();
91. jListUser.setModel(model);

94. }
96. //--------------------------//
98. public class IncomingReader implements Runnable
99. {
100. @Override
101. public void run()
102. {
103. String[] data;
104. String stream, done = "Done", connect = "Connect", disconnect = "Disconnect", chat = "Chat",userList="UserList";
105. try
106. {
107. while ((stream = reader.readLine()) != null) //read 1 line -> loop
108. {
109. data = stream.split(":");
110. if (data[2].equals(chat))
111. {
112. if(data[3].equals(tf\_username.getText())|| data[3].equals(tf\_toUser.getText())|| data[3].equals("ALL")){
113. if(data[0].equals(data[3])){ //fail connect&register
114. ta\_chat.append(data[1] +"\n");
115. ta\_chat.setCaretPosition(ta\_chat.getDocument().getLength()); // scroll last text position
116. tf\_username.setEditable(true);
117. tf\_password.setEditable(true);
118. b\_connect.setEnabled(true);
119. b\_register.setEnabled(true);
120. isConnected=false;
121. }else{
122. ta\_chat.append(data[0] +" to "+data[3]+ ": " + data[1] + "\n");
123. ta\_chat.setCaretPosition(ta\_chat.getDocument().getLength()); // scroll last text position
124. b\_connect.setEnabled(false);
125. b\_register.setEnabled(false);
126. }
127. }
128. }
129. else if (data[2].equals(connect))
130. {
131. ta\_chat.removeAll();
132. userAdd(data[0]);
133. }
134. else if (data[2].equals(disconnect))
135. {
136. userRemove(data[0]);
137. }
138. else if (data[2].equals(done))
139. {
140. //users.setText("");
141. writeUsers();
142. users.clear();
143. }
144. else if (data[2].equals(userList))
145. {
146. String str=data[1];
147. String[] JlistUser=str.split(",");
148. DefaultListModel model = new DefaultListModel();
149. jListUser.setModel(model);
151. for(int i=0;i<JlistUser.length;i++){
152. model.addElement(JlistUser[i]);
153. }
154. model.addElement("ALL");
155. }
156. }
157. }catch(Exception ex) { }
158. }
159. }
160. //--------------------------//
162. @SuppressWarnings("unchecked")
163. // <editor-fold defaultstate="collapsed" desc="Generated Code">
164. private void initComponents() {
165. lb\_port = new javax.swing.JLabel();
166. tf\_toUser = new javax.swing.JTextField();
167. lb\_username = new javax.swing.JLabel();
168. tf\_username = new javax.swing.JTextField();
169. lb\_password = new javax.swing.JLabel();
170. tf\_password = new javax.swing.JTextField();
171. b\_connect = new javax.swing.JButton();
172. b\_disconnect = new javax.swing.JButton();
173. jScrollPane1 = new javax.swing.JScrollPane();
174. ta\_chat = new javax.swing.JTextArea();
175. tf\_chat = new javax.swing.JTextField();
176. b\_send = new javax.swing.JButton();
177. livePanel = new javax.swing.JScrollPane();
178. jListUser = new javax.swing.JList<>();
179. jLabel2 = new javax.swing.JLabel();
180. b\_select = new javax.swing.JButton();
181. b\_register = new javax.swing.JButton();
182. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
183. setTitle("Chat - Client's frame");
184. setName("client"); // NOI18N
185. setResizable(false);
186. lb\_port.setText("to SomeOne :");
187. tf\_toUser.setEditable(false);
188. tf\_toUser.setHorizontalAlignment(javax.swing.JTextField.CENTER);
189. tf\_toUser.setText("ALL");
190. tf\_toUser.addActionListener(new java.awt.event.ActionListener() {
191. public void actionPerformed(java.awt.event.ActionEvent evt) {
192. tf\_toUserActionPerformed(evt);
193. }
194. });
195. lb\_username.setText("Username :");
196. tf\_username.addActionListener(new java.awt.event.ActionListener() {
197. public void actionPerformed(java.awt.event.ActionEvent evt) {
198. tf\_usernameActionPerformed(evt);
199. }
200. });
201. lb\_password.setText("Password : ");
202. tf\_password.addActionListener(new java.awt.event.ActionListener() {
203. public void actionPerformed(java.awt.event.ActionEvent evt) {
204. tf\_passwordActionPerformed(evt);
205. }
206. });
207. b\_connect.setText("Connect&Login");
208. b\_connect.addActionListener(new java.awt.event.ActionListener() {
209. public void actionPerformed(java.awt.event.ActionEvent evt) {
210. b\_connectActionPerformed(evt);
211. }
212. });
213. b\_disconnect.setText("Disconnect");
214. b\_disconnect.addActionListener(new java.awt.event.ActionListener() {
215. public void actionPerformed(java.awt.event.ActionEvent evt) {
216. b\_disconnectActionPerformed(evt);
217. }
218. });
219. ta\_chat.setColumns(20);
220. ta\_chat.setRows(5);
221. ta\_chat.setDisabledTextColor(new java.awt.Color(0, 0, 0));
222. ta\_chat.setEnabled(false);
223. jScrollPane1.setViewportView(ta\_chat);
224. b\_send.setText("SEND");
225. b\_send.addActionListener(new java.awt.event.ActionListener() {
226. public void actionPerformed(java.awt.event.ActionEvent evt) {
227. b\_sendActionPerformed(evt);
228. }
229. });
230. livePanel.setBackground(new java.awt.Color(255, 255, 255));
231. jListUser.setModel(new javax.swing.AbstractListModel<String>() {
232. String[] strings = { "Item 1", "Item 2", "Item 3", "Item 4", "Item 5" };
233. public int getSize() { return strings.length; }
234. public String getElementAt(int i) { return strings[i]; }
235. });
236. livePanel.setViewportView(jListUser);
237. jLabel2.setFont(new java.awt.Font("新細明體", 0, 14)); // NOI18N
238. jLabel2.setText("live user");
239. b\_select.setText("select");
240. b\_select.addActionListener(new java.awt.event.ActionListener() {
241. public void actionPerformed(java.awt.event.ActionEvent evt) {
242. b\_selectActionPerformed(evt);
243. }
244. });
245. b\_register.setText("Register");
246. b\_register.addActionListener(new java.awt.event.ActionListener() {
247. public void actionPerformed(java.awt.event.ActionEvent evt) {
248. b\_registerActionPerformed(evt);
249. }
250. });
251. javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
252. getContentPane().setLayout(layout);
253. layout.setHorizontalGroup(
254. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
255. .addGroup(layout.createSequentialGroup()
256. .addContainerGap()
257. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
258. .addGroup(layout.createSequentialGroup()
259. .addComponent(tf\_chat, javax.swing.GroupLayout.PREFERRED\_SIZE, 352, javax.swing.GroupLayout.PREFERRED\_SIZE)
260. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
261. .addComponent(b\_send, javax.swing.GroupLayout.DEFAULT\_SIZE, 85, Short.MAX\_VALUE))
262. .addComponent(jScrollPane1)
263. .addGroup(layout.createSequentialGroup()
264. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
265. .addComponent(lb\_password, javax.swing.GroupLayout.PREFERRED\_SIZE, 72, javax.swing.GroupLayout.PREFERRED\_SIZE)
266. .addComponent(lb\_username, javax.swing.GroupLayout.PREFERRED\_SIZE, 70, javax.swing.GroupLayout.PREFERRED\_SIZE))
267. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
268. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
269. .addComponent(tf\_username, javax.swing.GroupLayout.DEFAULT\_SIZE, 87, Short.MAX\_VALUE)
270. .addComponent(tf\_password))
271. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
272. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
273. .addGroup(layout.createSequentialGroup()
274. .addComponent(b\_disconnect, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
275. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
276. .addComponent(b\_register, javax.swing.GroupLayout.PREFERRED\_SIZE, 115, javax.swing.GroupLayout.PREFERRED\_SIZE))
277. .addComponent(b\_connect, javax.swing.GroupLayout.PREFERRED\_SIZE, 274, javax.swing.GroupLayout.PREFERRED\_SIZE))))
278. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 74, Short.MAX\_VALUE)
279. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
280. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
281. .addComponent(b\_select, javax.swing.GroupLayout.PREFERRED\_SIZE, 90, javax.swing.GroupLayout.PREFERRED\_SIZE)
282. .addGap(72, 72, 72))
283. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
284. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
285. .addGroup(layout.createSequentialGroup()
286. .addComponent(lb\_port)
287. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
288. .addComponent(tf\_toUser, javax.swing.GroupLayout.PREFERRED\_SIZE, 75, javax.swing.GroupLayout.PREFERRED\_SIZE))
289. .addComponent(livePanel, javax.swing.GroupLayout.PREFERRED\_SIZE, 143, javax.swing.GroupLayout.PREFERRED\_SIZE))
290. .addGap(48, 48, 48))
291. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
292. .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 56, javax.swing.GroupLayout.PREFERRED\_SIZE)
293. .addGap(93, 93, 93))))
294. );
295. layout.setVerticalGroup(
296. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
297. .addGroup(layout.createSequentialGroup()
298. .addContainerGap()
299. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
300. .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
301. .addComponent(lb\_username)
302. .addComponent(tf\_username)
303. .addComponent(b\_connect))
304. .addGroup(layout.createSequentialGroup()
305. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
306. .addComponent(lb\_port)
307. .addComponent(tf\_toUser, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))
308. .addGap(0, 0, Short.MAX\_VALUE)))
309. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
310. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
311. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
312. .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 23, javax.swing.GroupLayout.PREFERRED\_SIZE)
313. .addComponent(b\_disconnect, javax.swing.GroupLayout.PREFERRED\_SIZE, 23, javax.swing.GroupLayout.PREFERRED\_SIZE)
314. .addComponent(tf\_password, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)
315. .addComponent(b\_register))
316. .addComponent(lb\_password))
317. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
318. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
319. .addComponent(livePanel)
320. .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT\_SIZE, 310, Short.MAX\_VALUE))
321. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
322. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
323. .addComponent(tf\_chat)
324. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
325. .addComponent(b\_send, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
326. .addComponent(b\_select, javax.swing.GroupLayout.PREFERRED\_SIZE, 35, javax.swing.GroupLayout.PREFERRED\_SIZE)))
327. .addGap(23, 23, 23))
328. );
329. pack();
330. }// </editor-fold>
331. private void tf\_toUserActionPerformed(java.awt.event.ActionEvent evt) {
333. }
334. private void tf\_usernameActionPerformed(java.awt.event.ActionEvent evt) {
336. }
337. private void b\_connectActionPerformed(java.awt.event.ActionEvent evt) {
338. if (isConnected == false)
339. {
340. username = tf\_username.getText();
341. password = tf\_password.getText();
342. if(password.equals("")){
343. ta\_chat.append("username&password cannot be blank!!"+ "\n");
344. ta\_chat.setCaretPosition(ta\_chat.getDocument().getLength());
345. }else{
346. tf\_username.setEditable(false);
347. tf\_password.setEditable(false);
348. try
349. {
350. sock = new Socket(address, port);
351. InputStreamReader streamreader = new InputStreamReader(sock.getInputStream());
352. reader = new BufferedReader(streamreader); //set reader
353. writer = new PrintWriter(sock.getOutputStream()); //set output
354. writer.println(username + ":has connected.:Connect:"+password);
355. writer.flush(); //all send to server
356. isConnected = true;
357. }
358. catch (Exception ex)
359. {
360. ta\_chat.append("Cannot Connect! Try Again. \n");
361. tf\_username.setEditable(true);
362. }
363. ListenThread();
364. }
365. } else if (isConnected == true)
366. {
367. ta\_chat.append("You are already connected. \n");
368. }
369. }
370. private void b\_disconnectActionPerformed(java.awt.event.ActionEvent evt) {
371. sendDisconnect();
372. Disconnect();
373. DefaultListModel model = new DefaultListModel();
374. jListUser.setModel(model);
375. }
376. private void b\_sendActionPerformed(java.awt.event.ActionEvent evt) {
377. String nothing = "";
378. if ((tf\_chat.getText()).equals(nothing)) {
379. tf\_chat.setText("");
380. tf\_chat.requestFocus();
381. } else {
382. try {
383. writer.println(username + ":" + tf\_chat.getText() + ":" + "Chat"+":"+tf\_toUser.getText());
384. writer.flush(); // flushes the buffer
385. } catch (Exception ex) {
386. ta\_chat.append("Message was not sent. \n");
387. }
388. tf\_chat.setText("");
389. tf\_chat.requestFocus();
390. }
391. tf\_chat.setText("");
392. tf\_chat.requestFocus();
393. }
394. private void b\_selectActionPerformed(java.awt.event.ActionEvent evt) {
395. if(!jListUser.getSelectedValue().equals(tf\_username.getText())){
396. tf\_toUser.setText(jListUser.getSelectedValue());
397. }else{
398. ta\_chat.append("you can not send a message to yourself. \n");
399. tf\_chat.setText("");
400. tf\_chat.requestFocus();
401. }
403. }
404. private void tf\_passwordActionPerformed(java.awt.event.ActionEvent evt) {
405. // TODO add your handling code here:
406. }
407. private void b\_registerActionPerformed(java.awt.event.ActionEvent evt) {
408. if (isConnected == false)
409. {
410. username = tf\_username.getText();
411. password = tf\_password.getText();
412. if(password.equals("")){
413. ta\_chat.append("username&password cannot be blank!!"+ "\n");
414. ta\_chat.setCaretPosition(ta\_chat.getDocument().getLength());
415. }else{
416. tf\_username.setEditable(false);
417. tf\_password.setEditable(false);
418. try
419. {
420. sock = new Socket(address, port);
421. InputStreamReader streamreader = new InputStreamReader(sock.getInputStream());
422. reader = new BufferedReader(streamreader); //set reader
423. writer = new PrintWriter(sock.getOutputStream()); //set output
424. writer.println(username + ":has connected.:Register:"+password);
425. writer.flush(); //all send to server
426. isConnected = true;
427. }
428. catch (Exception ex)
429. {
430. ta\_chat.append("Cannot Register! Try Again. \n");
431. tf\_username.setEditable(true);
432. tf\_password.setEditable(true);
433. }
434. ListenThread();
435. }
436. }
437. }

440. public static void main(String args[])
441. {
442. java.awt.EventQueue.invokeLater(new Runnable()
443. {
444. @Override
445. public void run()
446. {
447. new client\_frame().setVisible(true);
448. }
449. });
450. }
452. // Variables declaration - do not modify
453. private javax.swing.JButton b\_connect;
454. private javax.swing.JButton b\_disconnect;
455. private javax.swing.JButton b\_register;
456. private javax.swing.JButton b\_select;
457. private javax.swing.JButton b\_send;
458. private javax.swing.JLabel jLabel2;
459. private javax.swing.JList<String> jListUser;
460. private javax.swing.JScrollPane jScrollPane1;
461. private javax.swing.JLabel lb\_password;
462. private javax.swing.JLabel lb\_port;
463. private javax.swing.JLabel lb\_username;
464. private javax.swing.JScrollPane livePanel;
465. private javax.swing.JTextArea ta\_chat;
466. private javax.swing.JTextField tf\_chat;
467. private javax.swing.JTextField tf\_password;
468. private javax.swing.JTextField tf\_toUser;
469. private javax.swing.JTextField tf\_username;
470. // End of variables declaration
471. }

伺服器端:

1. package chat\_server;
2. import java.io.\*;
3. import java.net.\*;
4. import java.util.\*;
5. public class server\_frame extends javax.swing.JFrame
6. {
7. ArrayList clientOutputStreams;
8. ArrayList<String> users;
9. ArrayList<String> account;
10. ArrayList<String> password;
12. public class ClientHandler implements Runnable
13. {
14. BufferedReader reader; //
15. Socket sock;
16. PrintWriter client;
17. public ClientHandler(Socket clientSocket, PrintWriter user)
18. {
19. client = user;
20. try
21. {
22. sock = clientSocket;
23. InputStreamReader isReader = new InputStreamReader(sock.getInputStream());
24. reader = new BufferedReader(isReader);
25. }
26. catch (Exception ex)
27. {
28. ta\_chat.append("Unexpected error... \n");
29. }
30. }
31. @Override
32. public void run()
33. {
34. String message, connect = "Connect", disconnect = "Disconnect", chat = "Chat" ,register="Register";
35. String[] data;
36. try
37. {
38. while ((message = reader.readLine()) != null)
39. {
40. ta\_chat.append("Received: " + message + "\n");
41. data = message.split(":");
43. for (String token:data)
44. {
45. ta\_chat.append(token + "\n");
46. }





53. if (data[2].equals(connect))
54. {
55. boolean check = false;
56. for(int i=0;i<account.size();i++){
57. if(account.get(i).equals(data[0])){
58. if(password.get(i).equals(data[3])){
59. tellEveryone((data[0] + ": "+data[0]+" has connected." + ":" + chat+":ALL"));
60. userAdd(data[0]);
61. tellLiveUser(users);
62. check=true;
63. }
64. }
65. }
66. if(check==false){
67. tellEveryone((data[0] + ": "+" your account and password are error." + ":" + chat+":"+data[0]));
68. }
70. }
71. else if (data[2].equals(disconnect))
72. {
73. tellEveryone((data[0] + ": "+data[0]+" has disconnected." + ":" + chat+":ALL"));
74. userRemove(data[0]);
75. tellLiveUser(users);
76. }
77. else if (data[2].equals(chat))
78. {
79. tellEveryone(message);
80. }
81. else if (data[2].equals(register))
82. {
83. boolean check = false;
84. for(int i=0;i<account.size();i++){
85. if(account.get(i).equals(data[0])){
86. tellEveryone((data[0] + ": "+" user name has been registered." + ":" + chat+":"+data[0]));
87. check=true;
88. }
89. }
90. if(check==false){
91. tellEveryone((data[0] + ": "+data[0]+" is a new user!!welcome." + ":" + chat+":ALL"));
92. account.add(data[0]);
93. password.add(data[3]);
94. userAdd(data[0]);
95. tellLiveUser(users);
96. }
97. }
98. else
99. {
100. ta\_chat.append("No Conditions were met. \n");
101. }
102. }
103. }
104. catch (Exception ex)
105. {
106. ta\_chat.append("Lost a connection. \n");
107. ex.printStackTrace();
108. clientOutputStreams.remove(client);
109. }
110. }
111. }
112. public server\_frame()
113. {
114. initComponents();
115. }
116. @SuppressWarnings("unchecked")
117. // <editor-fold defaultstate="collapsed" desc="Generated Code">
118. private void initComponents() {
119. jScrollPane1 = new javax.swing.JScrollPane();
120. ta\_chat = new javax.swing.JTextArea();
121. b\_start = new javax.swing.JButton();
122. b\_end = new javax.swing.JButton();
123. b\_users = new javax.swing.JButton();
124. b\_clear = new javax.swing.JButton();
125. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);
126. setTitle("Chat - Server's frame");
127. setName("server"); // NOI18N
128. setResizable(false);
129. ta\_chat.setColumns(20);
130. ta\_chat.setRows(5);
131. jScrollPane1.setViewportView(ta\_chat);
132. b\_start.setText("START");
133. b\_start.addActionListener(new java.awt.event.ActionListener() {
134. public void actionPerformed(java.awt.event.ActionEvent evt) {
135. b\_startActionPerformed(evt);
136. }
137. });
138. b\_end.setText("END");
139. b\_end.addActionListener(new java.awt.event.ActionListener() {
140. public void actionPerformed(java.awt.event.ActionEvent evt) {
141. b\_endActionPerformed(evt);
142. }
143. });
144. b\_users.setText("Online Users");
145. b\_users.addActionListener(new java.awt.event.ActionListener() {
146. public void actionPerformed(java.awt.event.ActionEvent evt) {
147. b\_usersActionPerformed(evt);
148. }
149. });
150. b\_clear.setText("Clear");
151. b\_clear.addActionListener(new java.awt.event.ActionListener() {
152. public void actionPerformed(java.awt.event.ActionEvent evt) {
153. b\_clearActionPerformed(evt);
154. }
155. });
156. javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
157. getContentPane().setLayout(layout);
158. layout.setHorizontalGroup(
159. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
160. .addGroup(layout.createSequentialGroup()
161. .addContainerGap()
162. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
163. .addComponent(jScrollPane1)
164. .addGroup(layout.createSequentialGroup()
165. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
166. .addComponent(b\_end, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
167. .addComponent(b\_start, javax.swing.GroupLayout.DEFAULT\_SIZE, 75, Short.MAX\_VALUE))
168. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 291, Short.MAX\_VALUE)
169. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
170. .addComponent(b\_clear, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)
171. .addComponent(b\_users, javax.swing.GroupLayout.DEFAULT\_SIZE, 103, Short.MAX\_VALUE))))
172. .addContainerGap())
173. );
174. layout.setVerticalGroup(
175. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
176. .addGroup(layout.createSequentialGroup()
177. .addContainerGap()
178. .addComponent(jScrollPane1, javax.swing.GroupLayout.DEFAULT\_SIZE, 340, Short.MAX\_VALUE)
179. .addGap(18, 18, 18)
180. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
181. .addComponent(b\_start)
182. .addComponent(b\_users))
183. .addGap(18, 18, 18)
184. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
185. .addComponent(b\_clear)
186. .addComponent(b\_end))
187. .addGap(21, 21, 21))
188. );
189. pack();
190. }// </editor-fold>
191. private void b\_endActionPerformed(java.awt.event.ActionEvent evt) {
192. try
193. {
194. Thread.sleep(5000); //5000 milliseconds is five second.
195. }
196. catch(InterruptedException ex) {Thread.currentThread().interrupt();}
198. tellEveryone("Server:is stopping and all users will be disconnected.\n:Chat");
199. ta\_chat.append("Server stopping... \n");
201. ta\_chat.setText("");
202. }
203. private void b\_startActionPerformed(java.awt.event.ActionEvent evt) {
204. Thread starter = new Thread(new ServerStart());
205. starter.start();
207. ta\_chat.append("Server started...\n");
208. }
209. private void b\_usersActionPerformed(java.awt.event.ActionEvent evt) {
210. ta\_chat.append("\n Online users : \n");
211. for (String current\_user : users)
212. {
213. ta\_chat.append(current\_user);
214. ta\_chat.append("\n");
215. }
216. tellLiveUser(users);
218. }
219. private void b\_clearActionPerformed(java.awt.event.ActionEvent evt) {
220. ta\_chat.setText("");
221. }
222. public static void main(String args[])
223. {
224. java.awt.EventQueue.invokeLater(new Runnable()
225. {
226. @Override
227. public void run() {
228. new server\_frame().setVisible(true);
230. }
231. });
232. }
234. public class ServerStart implements Runnable
235. {
236. @Override
237. public void run()
238. {
239. clientOutputStreams = new ArrayList();
240. users = new ArrayList();
241. account= new ArrayList();
242. password=new ArrayList();
243. account.add("123");
244. password.add("456");
245. account.add("789");
246. password.add("456");
247. try
248. {
249. ServerSocket serverSock = new ServerSocket(2222);
250. while (true)
251. {
252. Socket clientSock = serverSock.accept();
253. PrintWriter writer = new PrintWriter(clientSock.getOutputStream());
254. clientOutputStreams.add(writer);
255. Thread listener = new Thread(new ClientHandler(clientSock, writer));
256. listener.start();
257. ta\_chat.append("Got a connection. \n");
258. }
259. }
260. catch (Exception ex)
261. {
262. ta\_chat.append("Error making a connection. \n");
263. }
264. }
265. }
267. public void userAdd (String data)
268. {
269. String message, add = ": :Connect", done = "Server: :Done", name = data;
270. ta\_chat.append("Before " + name + " added. \n");
271. users.add(name);
272. ta\_chat.append("After " + name + " added. \n");
273. String[] tempList = new String[(users.size())];
274. users.toArray(tempList);
275. for (String token:tempList)
276. {
277. message = (token + add);
278. tellEveryone(message);
279. }
280. tellEveryone(done);
281. }
283. public void userRemove (String data)
284. {
285. String message, add = ": :Connect", done = "Server: :Done", name = data;
286. users.remove(name);
287. String[] tempList = new String[(users.size())];
288. users.toArray(tempList);
289. for (String token:tempList)
290. {
291. message = (token + add);
292. tellEveryone(message);
293. }
294. tellEveryone(done);
295. }
297. public void tellEveryone(String message)
298. {
299. Iterator it = clientOutputStreams.iterator();
300. while (it.hasNext())
301. {
302. try
303. {
305. PrintWriter writer = (PrintWriter) it.next();
306. writer.println(message);
307. ta\_chat.append("Sending: " + message +"\n");
308. writer.flush();
309. ta\_chat.setCaretPosition(ta\_chat.getDocument().getLength());
310. }
311. catch (Exception ex)
312. {
313. ta\_chat.append("Error telling everyone. \n");
314. }
315. }
316. }
318. public void tellLiveUser(ArrayList allUser){
319. String userList="";
320. for (String current\_user : users)
321. {
322. userList=userList+current\_user+",";
323. }
324. if(userList.length()!=0){
325. userList=userList.substring(0,userList.length()-1);
326. }
327. tellEveryone(("server:" + userList + ":" + "UserList"));
328. System.out.println(userList);
330. }
332. // Variables declaration - do not modify
333. private javax.swing.JButton b\_clear;
334. private javax.swing.JButton b\_end;
335. private javax.swing.JButton b\_start;
336. private javax.swing.JButton b\_users;
337. private javax.swing.JScrollPane jScrollPane1;
338. private javax.swing.JTextArea ta\_chat;
339. // End of variables declaration
340. }
341. 參考資料: 若有參考教科書以外的書籍或網路上的資料, 請在此部份詳列

大架構:<https://www.youtube.com/watch?v=hZgntu7889Q>

參考socket連線方法、建立伺服器、傳遞訊息給所有連伺服器的使用者的方法

、伺服器顯示有誰連線