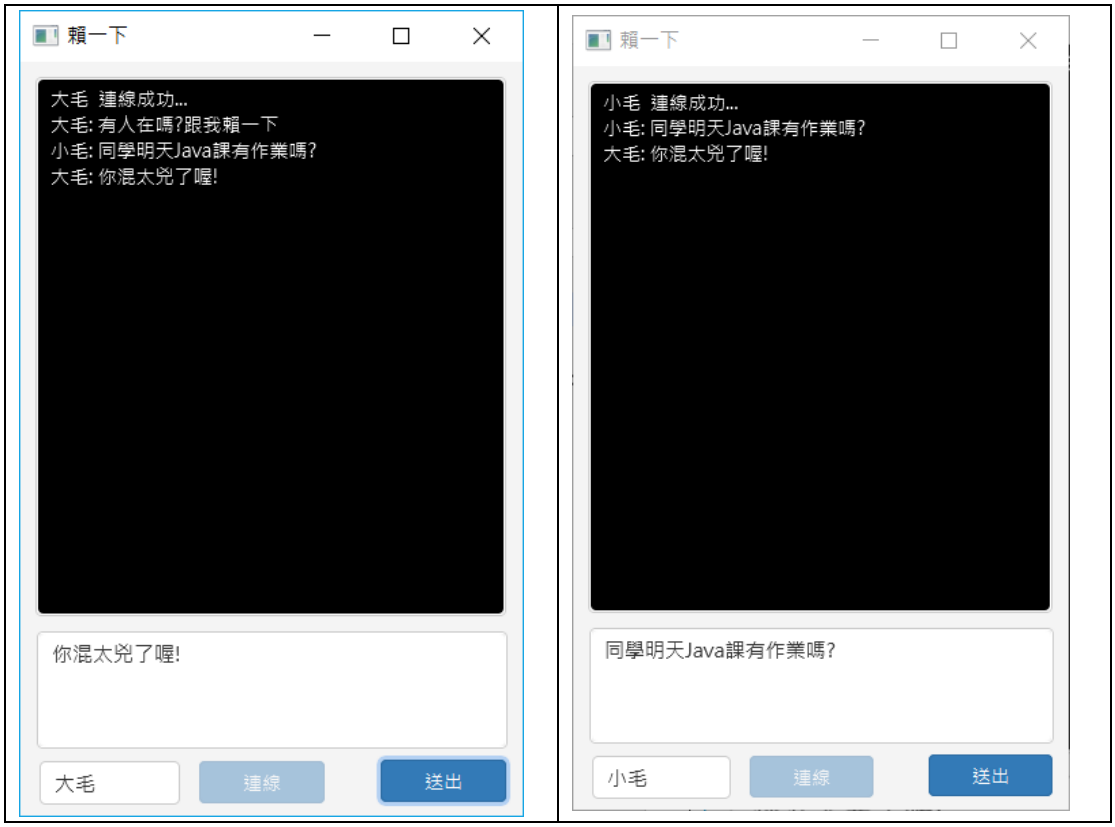
我也會 Line 喔!

網路(Network)應用程式:賴一下!

只要在網路傳送訊息、檔案，都會用到 Java 的 net 套件(FTP, P2P,網路通訊 …)。



這裡介紹的網路應用程式—多人聊天室，不是很簡單，你需要具備進階 Java 程式的能力：

- GUI 設計
- 自訂事件方法
- 多執行緒
- 例外處理
- 網路 Socket
- ArrayList< >或 HashSet< > 進階資料結構的使用

基本練功:server, client

Server 端: ServerSocket
ServerSocket.listen()監聽是否有連線進來
ServerSocket.accept()取得 client 端連線的 Socket 物件

Client 端: Socket

當 Server 和 Client 連接後:
準備兩個水管：
in 進來的
out 出去的

接收進來的對方訊息:
in.readLine()

送出訊息給對方
out.println()

伺服器

```
package chapter31_chatting_tutorial;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class MyServer {
	public static void main(String[] args) throws IOException {
```

```
        //伺服器初始化
        ServerSocket server = new ServerSocket(1024);

        //等待連線...
        System.out.println("等待連線...");
        Socket socket = server.accept();

        //建立串流 一個讀進來 一個寫出去
        DataInputStream fromClient = new DataInputStream(socket.getInputStream());
        DataOutputStream toClient = new DataOutputStream(socket.getOutputStream());

        //讀訊息  伺服器 read 讀取次數必須與 client 配合
        //讀不到資料時會天荒地老等 read 下去 甚至會打死結
        String msg = fromClient.readUTF();
        System.out.println(msg);

        //送出訊息
        String response = String.format("伺服器回應:%s", msg);
        toClient.writeUTF(response);
    }
}
```

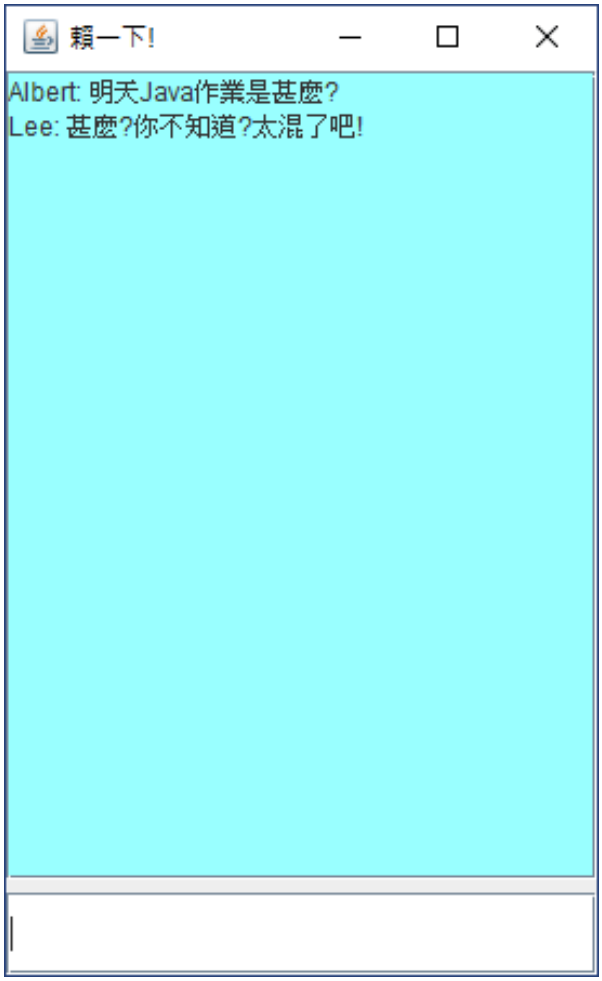
Client 端

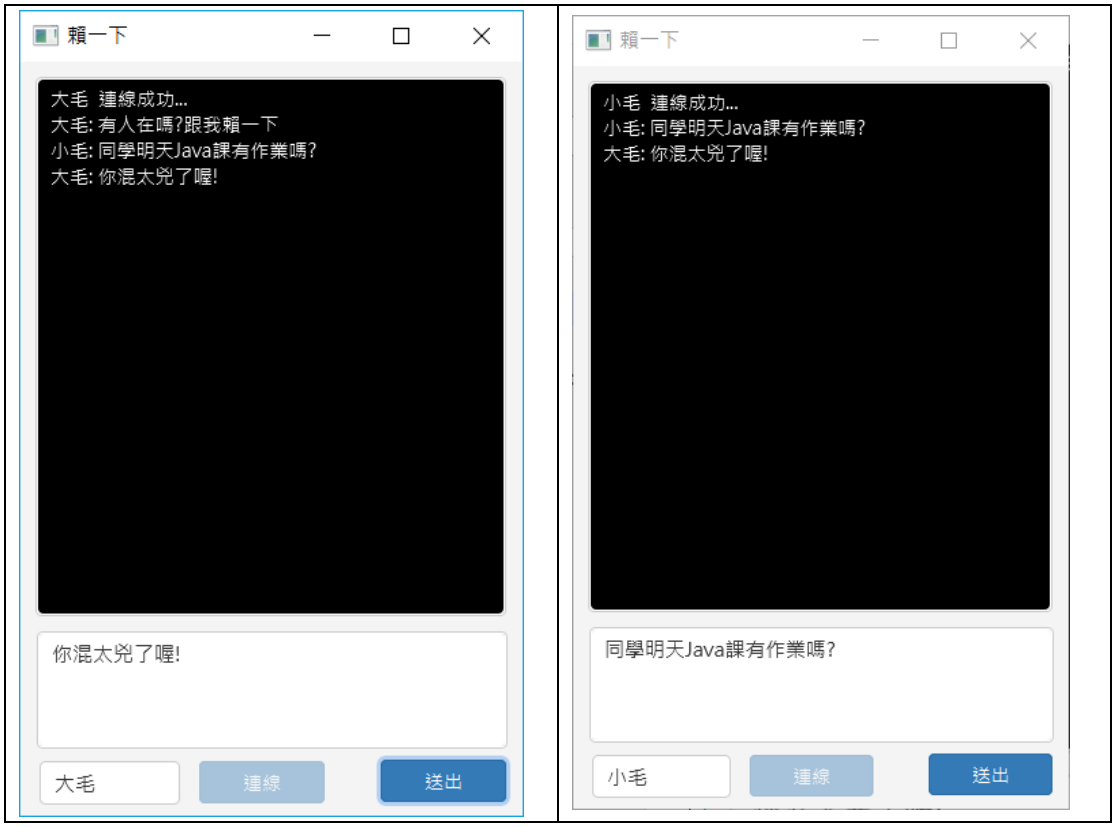
```
package chapter31_chatting_tutorial;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

public class MyClient {

    public static void main(String[] args) throws IOException {
```

```
        //建立連線
        Socket socket = new Socket("localhost", 1024);

        //建立串流 一個讀進來 一個寫出去
        DataInputStream fromServer = new DataInputStream(socket.getInputStream());
        DataOutputStream toServer = new DataOutputStream(socket.getOutputStream());

        //送出訊息
        toServer.writeUTF("李大同");

        //讀入
        System.out.println(fromServer.readUTF());
    }
}
```

目標:完成一個簡單的聊天室程式。

會寫這個程式可以去 Line 工作嗎?


賴一下!
Albert: 明天Java作業是甚麼?
Lee: 甚麼?你不知道?太混了吧!



Lab#0: 無功能畫面板程式

先把老師給的只有畫面版的 Client.java, Server.java 準備好，讓我們一步一步分解動作，完成複雜的任務。

Client 端(手工版，沒有用 FXML 設計)

Lab#1: Client 送出使用者名稱與接收伺服器回應訊息各一次

```
btnConnect.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {

        try {
            //產生一個Socket物件-連線到伺服器
            Socket socket = new Socket(server_ip, 1024);
            fromServer = new DataInputStream(socket.getInputStream());
            toServer = new DataOutputStream(socket.getOutputStream());

            //送出使用者名稱給伺服器
            String user_name = user.getText();

            toServer.writeUTF(user_name);

            //按鈕狀態 與 顯示
            display.appendText(user_name + "送出使用者名稱給伺服器\n");
            btnConnect.setDisable(true);
            btnSubmit.setDisable(false);

            //讀入一次伺服器送過來的資訊
            String msg = "";

            msg = fromServer.readUTF();
            display.appendText(msg + "\n");

            //---小小的實驗測試1:
            //再讀入一次伺服器送過來的資訊
            //(此時伺服器並沒有送訊息過來)
            //是不是會一直等下去?視窗畫面會停滯住?
            //何時會繼續執行?等到伺服器送訊息過來才會繼續執行下去
            //msg = fromServer.readUTF();
            //display.appendText(msg + "\n");
            //---小小的實驗測試2:
            //連續監聽伺服器串流通道訊息
            //是不是會一直等下去?視窗畫面會停滯住?永遠停滯!!
            while (true) {
                //讀入伺服器送過來的資訊
                msg = fromServer.readUTF();
                display.appendText(msg + "\n");
            }

        } catch (IOException e) {
            System.out.println("無法連線:" + e.toString());
        }
    } //handle()
}); //btnConnect setOnAction事件
```

Lab#2: Client 連續監聽接收訊息—視窗被凍結住(卡

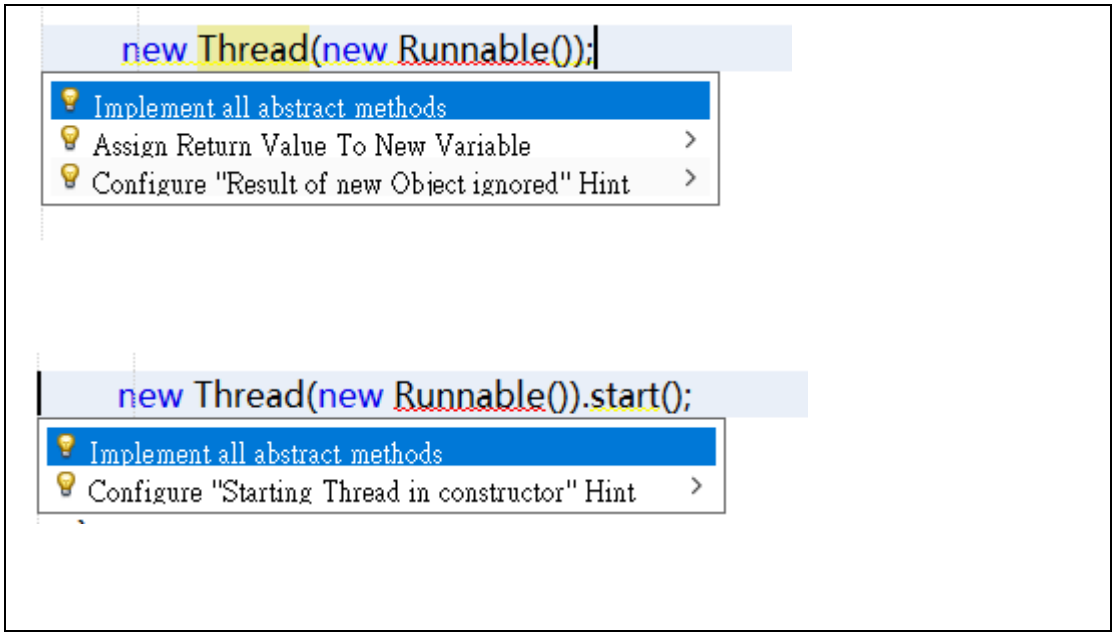
住)

程式碼參看前一個。

Lab#3: Client 連續監聽接收訊息—執行緒出馬，視窗不再被凍結

不能在 JavaFX 應用程式執行緒上執行長時間執行的操作，因為這會讓 JavaFX 更新 UI 被卡住，導致凍結的 UI。

新增空的執行緒類別



```
new Thread(new Runnable() {
    @Override
    public void run() {


    }
}).start();
```

將程式寫在 run(){}方法之內:

```
btnConnect.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {

        //匿名方式產新一個新執行緒物件
        //送出使用者名稱並連續監聽伺服器串流通道訊息
        //此處一定要用執行緒，否則會卡住
        //因為while是個無窮迴圈，不會將控制權交出給按鈕事件
        new Thread(new Runnable() {
            @Override
            public void run() {

                try {
                    //產生一個Socket物件-連線到伺服器
                    Socket socket = new Socket(server_ip, 1024);
                    fromServer = new DataInputStream(socket.getInputStream());
                    toServer = new DataOutputStream(socket.getOutputStream());

                    //送出使用者名稱給伺服器
                    String user_name = user.getText();

                    toServer.writeUTF(user_name);

                    //按鈕狀態 與 顯示
                    display.appendText(user_name + "送出使用者名稱給伺服器\n");
                    btnConnect.setDisable(true);
                    btnSubmit.setDisable(false);
```

```
                    //讀入一次伺服器送過來的歡迎詞訊息
                    String msg = "";

                    msg = fromServer.readUTF();
                    display.appendText(msg + "\n");

                    //連續監聽伺服器串流通道訊息
                    //是不是會一直等下去?視窗畫面會停滯住?不會停滯!!
                    while (true) {
                        //讀入伺服器送過來的資訊
                        msg = fromServer.readUTF();
                        display.appendText(msg + "\n");
                    }

                } catch (IOException e) {
                    System.out.println("無法連線:" + e.toString());
                }

            }
        }).start();

    } //handle()
}); //btnConnect setOnAction事件
```

Lab#4:定義送出留言按鈕事件

```
btnSubmit.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        try {
            //送出留言訊息給伺服器
            toServer.writeUTF(input.getText());
            toServer.flush();
        } catch (IOException e) {
            display.appendText("傳送訊息發生異常(斷線)\n");
            System.out.println("傳送訊息發生異常:" + e.toString());
        }

    }
});
```

Lab#4.5:定義送出留言按鈕事件—視窗元件出現異常

toServer.writeUTF(input.getText());

display.appendText("傳送一筆訊息給伺服器\n");

上面這行會有異常：java.lang.ArrayIndexOutOfBoundsException: -1

原因: toServer 與 display 分屬不同的執行緒，個別運作不會產生問題，同時一起運作會產生異常。

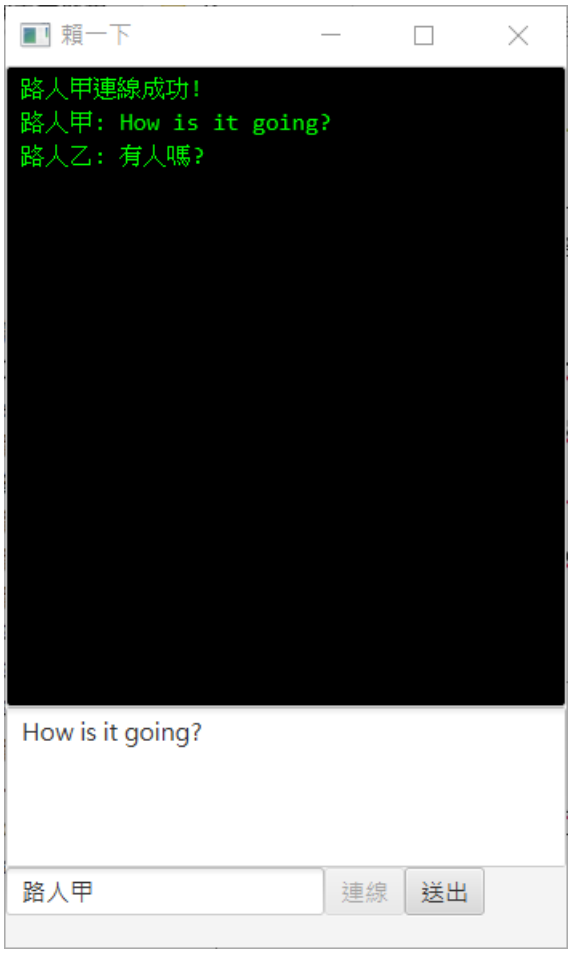
方式 1: 改用以下方式執行:Platform.runLater 去更新 UI 元件

方式 2: 整段程式碼用執行緒方式執行

```
btnSubmit.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {

        //方式 1:
        try {
            toServer.writeUTF(input.getText());
            toServer.flush();
            System.out.println("執行 1:完成傳送一筆訊息給伺服器");//這一行可以正常運作

            display.appendText("傳送一筆訊息給伺服器\n");
            //上面這行會有異常：java.lang.ArrayIndexOutOfBoundsException: -1
            //原因: toServer 與 display 分屬不同的執行緒，個別運作不會產生問題，同時一起運作會產生異常
            //方式 1:改用以下方式執行:Platform.runLater 去更新 UI 元件
            //方式 2: 執行緒方式執行
            Platform.runLater(new Runnable() {
                @Override
                public void run() {
                    display.appendText("傳送一筆訊息給伺服器\n");
                    System.out.println("執行 2:之後才會執行 runLater()更新 UI");
                }
            });

        } catch (IOException e) {
            display.appendText("可能伺服器斷線，無法送出!\n");
        }
```

```
                /*
                //方式 2: 執行緒方式執行
                new Thread(new Runnable() {
                    @Override
                    public void run() {
                        System.out.println("執行 0:Thread run()開始");
                        try {
                            toServer.writeUTF(input.getText());
                            //toServer.flush();
                            System.out.println("執行 1:完成傳送一筆訊息給伺服器");//這一行可以正常運作

                            //display.appendText("傳送一筆訊息給伺服器\n");
                            //上面這行會有異常：java.lang.ArrayIndexOutOfBoundsException: -1
                            //原因: toServer 與 display 分屬不同的執行緒，個別運作不會產生問題，同時一起運作會產生異常
                            //方式 1:改用以下方式執行:Platform.runLater 去更新 UI 元件
                            //方式 2: 執行緒方式執行
                            display.appendText("傳送一筆訊息給伺服器\n");
                            System.out.println("執行 2:執行 runLater()更新 UI");
                        } catch (IOException e) {
                            display.appendText("可能伺服器斷線，無法送出!\n");
                        }
                        System.out.println("執行 3:Thread run()結束");
                    }//run()
                }).start(); */

                System.out.println("執行 4:先完成按鈕事件 handle()");
            } //handle()
        }); //button setOnAction()
```

Lab#5: Client 完整功能:使用 Thread 類別




```
package chapter31_chatting_tutorial;


import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

import javafx.application.Application;

import javafx.application.Platform;

import javafx.event.ActionEvent;

import javafx.event.EventHandler;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.TextArea;

import javafx.scene.control.TextField;

import javafx.scene.layout.FlowPane;

import javafx.stage.Stage;
```

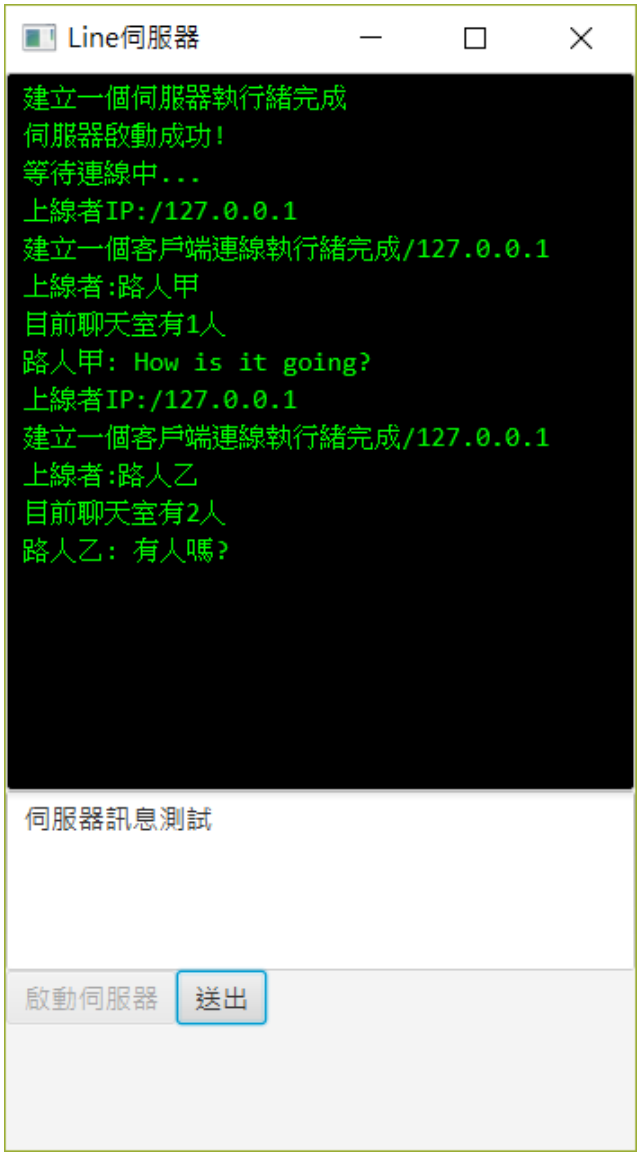
```
public class V20_Client_Compelete extends Application {

    TextArea display = new TextArea();
    TextArea input = new TextArea("How is it going?");
    TextField user = new TextField("路人甲");
    Button btnConnect = new Button("連線");
    Button btnSubmit = new Button("送出");

    //有兩個按鈕事件會用到輸入輸出串流
    DataOutputStream toServer;
    DataInputStream fromServer;

    //建構子
    //建構子會優先執行，之後再執行 public void start(Stage primaryStage){}
    public V20_Client_Compelete() {

    }

    @Override
    public void start(Stage primaryStage) {
        FlowPane root = new FlowPane();
        display.setPrefSize(350, 400);
        input.setPrefSize(350, 100);
        display.setStyle(""
                + "-fx-control-inner-background:#000000; "
                + "-fx-font-family: Consolas; "
                + "-fx-highlight-fill: #00ff00; "
                + "-fx-highlight-text-fill: #000000; "
                + "-fx-text-fill: #00ff00; ");

        //texArea 自動換行
        display.setWrapText(true);
        input.setWrapText(true);

        btnSubmit.setDisable(true);

        root.getChildren().add(display);
```

```
        root.getChildren().add(input);
        root.getChildren().add(user);
        root.getChildren().add(btnConnect);
        root.getChildren().add(btnSubmit);

        Scene scene = new Scene(root, 350, 550);

        primaryStage.setTitle("賴一下");
        primaryStage.setScene(scene);
        primaryStage.show();
        primaryStage.setOnCloseRequest(e -> {
            Platform.exit();
            System.exit(0);
        });

        btnSubmit.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {

                try {
                    //送出訊息給伺服器
                    toServer.writeUTF(input.getText());
                    //toServer.flush();
                } catch (IOException e) {
                    System.out.println("傳送訊息發生異常:" + e.toString());
                    display.appendText("傳送訊息發生異常(斷線)\n");
                }

            }
        });

        btnConnect.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {

                //匿名方式產新一個新執行緒物件
                //此處一定要用執行緒，否則會卡住
                new ClientConnect().start();
```

```
                System.out.println("建立一個 Client 連線執行緒完成");


            } //handle(){    }
        });    //btn.setOnAction( {      }    );
    } //public void start(Stage stage)結束

    public static void main(String[] args) {
        launch(args);
    }

    //ClientConnect 內部類別較為方便，因為要用到前面定義的全域變數
    //送出使用者名稱並連續監聽伺服器串流通道訊息
    class ClientConnect extends Thread {

        //因為 while 是個無窮迴圈，不會將控制權交出給按鈕事件
        @Override
        public void run() {

            try {
                //產生一個 Socket 物件-連線到伺服器
                Socket socket = new Socket("localhost", 1024);
                fromServer = new DataInputStream(socket.getInputStream());
                toServer = new DataOutputStream(socket.getOutputStream());

                //送出使用者名稱給伺服器
                String user_name = user.getText();

                toServer.writeUTF(user_name);

                //
                display.appendText(user_name + "連線成功!\n");
                btnConnect.setDisable(true);
                btnSubmit.setDisable(false);

                //連續監聽伺服器串流通道訊息
                while (true) {
                    //讀入伺服器送過來的資訊
```

```
                        String msg = fromServer.readUTF();
                        display.appendText(msg + "\n");
                    }

                } catch (IOException e) {
                    display.appendText("無法連線\n");
                    System.out.println("無法連線:" + e.toString());
                }
            } //run()

        } //ClientConnect

}
```

Lab#6: Server 端(較為複雜)




```
while (true)    外部迴圈
{
  連續監聽所有的 clients 連線(永遠等下去，等到天荒地老)
  等到有人來建立連線，進入內部迴圈
     while (true)      內部迴圈
       {
           持續監聽接受來自於 client 送來的訊息(永遠等下去，等到天荒地老)
       }
}
```

兩階段使用執行緒

外部第 1 個執行緒(ServerStart 執行緒，只有 1 個，將所有連線程式碼包起來): 啟動伺服器，用一個(外部)迴圈連續監聽所有的 clients 一個一個進來連線(等待接聽每個人，永遠等下去，等到天荒地老)

終於等到有人來建立連線，立刻替進來的使用者初始化一個 ClientStart 執行緒物件。對每一個 client 產生其各自的執行緒。外部迴圈產生 ClientStart 執行緒: 第 1 個、第 2 個、...、第 n-1 個、第 n 個...

ClientStart 執行緒內部:while 迴圈持續監聽接受來自於伺服器的訊息(永遠等著伺服器，等到天荒地老)

外部第 1 個執行緒(只有 1 個)

new ClientConnect(socket).start();

```
//---建立一個伺服器 ServerSocket
ServerSocket server = new ServerSocket(port);

//無窮迴圈監聽誰進來連線
while (true) {
    //(1)等待有 client 連線，
    // 天荒地老地等下去直到有人連線，才會跳到下一行程式
    Socket socket = server.accept();

    //(2)終於等到有人來建立連線之後，
    // 初始化一個 client 執行緒物件，
    // 在執行緒裡面做詳細的訊息處理
    // 若連線有 3 人，就會有 3 個 client 執行緒物件被產生
    // 若沒有使用 Thread，則控制權會等待卡在無窮迴圈裡，造成視窗無回應
    new ClientConnect(socket).start();

}//外層 while
```

第一個執行緒的迴圈:產生第 1 個、第 2 個、...、第 n-1 個、第 n 個執行緒

new ClientConnect(socket).start();

```
// 建立輸出與輸入串流
//讀取一次，取得姓名
//把串流通道存放在連線集合中
//無窮迴圈 接收進來的訊息 並廣播出去
// 若沒有執行緒第 2 個人連線進來會卡住(因為在迴圈內等待)
while (true) {
      //讀取使用者送來的訊息
      String msg = fromClient.readUTF();
      //送出訊息給所有的通道
      for (DataOutputStream writer : output2clients) {
            writer.writeUTF(String.format("%s: %s", name, msg));
      }
 }//內層 while 結束
```

Server.java

```
package chapter31_chatting_tutorial;


import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.layout.FlowPane;
import javafx.stage.Stage;


public class V20_Server_Compelete extends Application {


    // 全域變數 整個程式有多個地方會用這些變數
    private final TextArea display = new TextArea();
    private final Button btnSubmit = new Button("送出");
    private final Button btnConnect = new Button("啟動伺服器");
    private final TextArea input = new TextArea("伺服器訊息測試");
    private final int port = 1024;
```

```
//存放所有連線進來的 soket
private final List<DataOutputStream> output2clients = new ArrayList();

public V20_Server_Compelete() {

}

@Override
public void start(Stage stage) {

    FlowPane root = new FlowPane();
    display.setPrefSize(350, 400);
    input.setPrefSize(350, 100);
    btnSubmit.setDisable(true);

    //texArea 自動換行
    display.setWrapText(true);
    input.setWrapText(true);

    root.getChildren().add(display);
    root.getChildren().add(input);
    root.getChildren().add(btnConnect);
    root.getChildren().add(btnSubmit);

    Scene scene = new Scene(root, 350, 600);

    stage.setTitle("Line 伺服器");
    stage.setScene(scene);
    stage.show();
    stage.setOnCloseRequest(ex -> {
        System.exit(0); //結束這個視窗與這個程式的所有執行緒
    });

    //設定 css 樣式
    display.setStyle(""
            + "-fx-control-inner-background:#000000; "
            + "-fx-font-family: Consolas; "
            + "-fx-highlight-fill: #00ff00; "
```

```
                + "-fx-highlight-text-fill: #000000; "
                + "-fx-text-fill: #00ff00; ");

        btnSubmit.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                String msg = String.format("server:%s", input.getText());
                display.appendText(msg + "\n");

                //送出訊息給所有的通道
                try {
                    for (DataOutputStream writer : output2clients) {
                        writer.writeUTF(msg);
                    }
                } catch (IOException e) {
                    display.appendText("無法送出訊息");
                }
            }
        });

        btnConnect.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {

                //----啟動伺服器，連續監聽所有的 clients-----------
                // 不能寫在建構子 因為是按鈕事件驅動的，須等按鈕初始化之後(start()是在建構子之後執行)
                // 若沒有使用 Thread，則控制權會等待卡在無窮迴圈裡，造成視窗無回應
                new ServerStart().start();

                display.appendText("建立一個伺服器執行緒完成\n");
                //----啟動伺服器程式結束----------------------

            }//handle
        }); //btn action 結束

    } //public void start(Stage stage)結束

    public static void main(String[] args) {
```

```
        launch(args);
    }

    //寫成內部類別較為方便，因為要用到前面定義的全域變數
    class ServerStart extends Thread {

        //全域變數宣告區
        @Override
        public void run() {

            //System.out.println("here");
            // try 區塊 1----------------
            try {

                //---建立一個伺服器 ServerSocket
                ServerSocket server = new ServerSocket(port);
                display.appendText("伺服器啟動成功!\n");
                display.appendText("等待連線中...\n");
                btnConnect.setDisable(true);
                btnSubmit.setDisable(false);

                //無窮迴圈監聽誰進來連線
                while (true) {

                    //(1)等待有 client 連線，
                    // 天荒地老地等下去直到有人連線，才會跳到下一行程式
                    Socket socket = server.accept();
                    display.appendText("上線者 IP:" + socket.getInetAddress() + "\n");

                    //(2)終於等到有人來建立連線之後，
                    // 初始化一個 client 執行緒物件，
                    // 在執行緒裡面做詳細的訊息處理
                    // 若連線有 3 人，就會有 3 個 client 執行緒物件被產生
                    // 若沒有使用 Thread，則控制權會等待卡在無窮迴圈裡，造成視窗無回應
                    new ClientConnect(socket).start();
                    display.appendText("建立一個客戶端連線執行緒完成" + socket.getInetAddress() + "\n");
                    System.out.println("建立一個客戶端連線執行緒完成" + socket.getInetAddress());
```

```
                }//最外層 while
                // try 區塊 1 結束----------
            } catch (IOException e) {
                display.appendText("建立伺服器異常\n");
                display.appendText(e.toString() + "\n");
                //System.out.println(e.toString());
            }
            //try-catch 結束----------
        } //run()結束

} //ServerStart

//ClientConnect 內部類別較為方便，因為要用到前面定義的全域變數
class ClientConnect extends Thread {

    //這幾個變數是 Thread 裡面的全域變數
    //每個 client 都各自有一套，沒有重複
    private DataOutputStream toClient;
    private String name;
    private Socket socket;

    public ClientConnect(Socket socket)
    {
        this.socket = socket;
    }

    @Override
    public void run() {

        try {
            // try 區塊 2----------
            // 建立輸出與輸入串流
            DataInputStream fromClient = new DataInputStream(socket.getInputStream());
            toClient = new DataOutputStream(socket.getOutputStream());

            //讀取一次，取得姓名
            name = fromClient.readUTF();
            display.appendText("上線者:" + name + "\n");
```

```
                //把串流通道存放在連線集合中
                output2clients.add(toClient);
                display.appendText("目前聊天室有" + output2clients.size() + "人\n");

                //無窮迴圈 接收進來的訊息 並廣播出去
                // 若沒有執行緒第 2 個人連線進來會卡住(因為在迴圈內等待)
                //直到使用者結束連線，會跳去執行 finally 區塊
                while (true) {

                    //讀取使用者送來的訊息
                    String msg = fromClient.readUTF();

                    //使用者訊息很多，輸出到 terminal，或存到 log 檔
                    display.appendText(String.format("%s: %s\n", name, msg));
                    //System.out.printf("%s: %s\n", name, msg);

                    //送出訊息給 client 只有回應給一個人，這不是我們要的功能
                    //toClient.writeUTF(String.format("%s: %s", name, msg));
                    //送出訊息給所有的通道  這才是我們要的功能
                    for (DataOutputStream writer : output2clients) {
                        writer.writeUTF(String.format("%s: %s", name, msg));
                        //writer.flush();
                    }
                }//內層 while 結束
                // try 區塊 2 結束----------
            } catch (IOException e) {
                display.appendText("client 結束通訊:" + socket.getInetAddress() + "\n");
                display.appendText("client 結束通訊:" + name + "\n");

                output2clients.remove(toClient);//移除 toClient 串流
                display.appendText("目前聊天室有" + output2clients.size() + "人\n");
            }

        } //run()
    } //ClientConnect
}
```

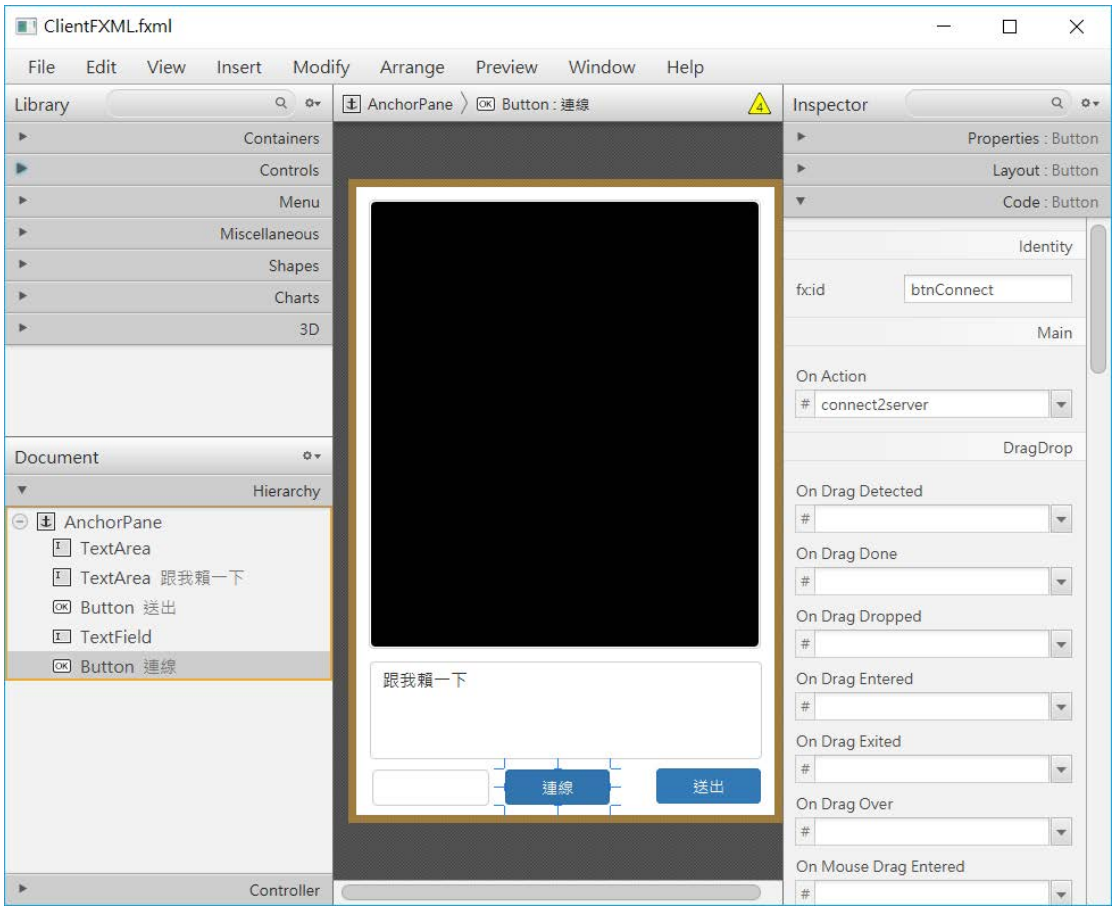
Lab#7: Client 端 FXML 美觀版

ClientFXMLMain.java

```
package chapter31_chatting_tutorial;

import java.io.IOException;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class ClientFXMLMain extends Application {

	@Override
	public void start(Stage primaryStage) throws IOException {

		Parent root = FXMLLoader.load(this.getClass().getResource("ClientFXML.fxml"));

		Scene scene = new Scene(root);

		primaryStage.setTitle("賴一下");
		primaryStage.setScene(scene);
		primaryStage.show();
		primaryStage.setOnCloseRequest(e -> {
			Platform.exit();
			System.exit(0);
		});
	}

	public static void main(String[] args) {
		launch(args);
```

```
    }


}
```

ClientFXML.fxml




```
<?xml version="1.0" encoding="UTF-8"?>


<?import java.lang.*?>
<?import java.util.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>


<AnchorPane id="AnchorPane" prefHeight="552.0" prefWidth="368.0" stylesheets="@bootstrap3.css" xmlns="http://javafx.com/javafx/8" xmlns:fx="http://javafx.com/fxml/1"
```

```
fx:controller="chapter31_chatting_tutorial.ClientFXMLController">

    <children>

        <TextArea fx:id="display" editable="false" layoutX="11.0" layoutY="11.0" prefHeight="395.0" prefWidth="345.0" stylesheets="@myTextArea.css" wrapText="true" />

        <TextArea fx:id="input" layoutX="12.0" layoutY="417.0" prefHeight="86.0" prefWidth="345.0" text="跟我頻一下" wrapText="true" />

        <Button fx:id="btnSubmit" layoutX="264.0" layoutY="511.0" mnemonicParsing="false" onAction="#send" prefHeight="45.0" prefWidth="92.0" styleClass="primary" text="送出" />

        <TextField fx:id="textfield_name" layoutX="14.0" layoutY="512.0" prefHeight="32.0" prefWidth="103.0" />

      <Button fx:id="btnConnect" layoutX="131.0" layoutY="512.0" mnemonicParsing="false" onAction="#connect2server" prefHeight="45.0" prefWidth="92.0" styleClass="primary" text="連線" />

    </children>
</AnchorPane>
```

ClientFXMLController.java

```
package chapter31_chatting_tutorial;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;

public class ClientFXMLController implements Initializable {

    @FXML
    private TextArea display;
    @FXML
    private TextArea input;
    @FXML
    private Button btnSubmit;
```

```
@FXML
TextField textfield_name;

@FXML
private Button btnConnect;

private final int port = 1024;
private final String server_ip = "localhost";

//有兩個按鈕事件會用到輸入輸出串流
DataOutputStream toServer;
DataInputStream fromServer;

@Override
public void initialize(URL url, ResourceBundle rb) {
    //先把送出訊息按鈕 disable
    btnSubmit.setDisable(true);
}

@FXML
private void send(ActionEvent event) {

    try {
        //送出訊息給伺服器
        toServer.writeUTF(input.getText());
        toServer.flush();
    } catch (IOException e) {
        System.out.println("傳送訊息發生異常:" + e.toString());
        display.appendText("傳送訊息發生異常(斷線)\n");
    }
}

@FXML
private void connect2server(ActionEvent event) {

    if (textfield_name.getText().isEmpty()) {
        display.appendText("請輸入使用者代號...\n");
```

```
            return;
        }

        //匿名方式產新一個新執行緒物件
        new Thread(new Runnable() {
            @Override
            public void run() {

                try {
                    Socket socket = new Socket("localhost", 1024);
                    fromServer = new DataInputStream(socket.getInputStream());
                    toServer = new DataOutputStream(socket.getOutputStream());
                    display.setText(textfield_name.getText() + "    連線成功...\n");
                    //連線之後將連線按鈕 disable
                    btnConnect.setDisable(true);
                    btnSubmit.setDisable(false);

                    //送出使用者名稱給伺服器
                    toServer.writeUTF(textfield_name.getText());

                    //連續監聽伺服器串流通道訊息
                    while (true) {
                        //讀入伺服器送過來的資訊
                        String msg = fromServer.readUTF();
                        display.appendText(msg + "\n");
                    }

                } catch (IOException e) {
                    display.setText("無法連線...\n");
                    System.out.println("無法連線:" + e.toString());
                }
            } //run()
        }).start();
    } //connect2server
}//class end
```