

# PROJECT: Java Chat Server

---

Name: HuChenkai ID: 3210103296

## 1 BASICS

---

### 1.1 Requirements

Implement a multi-client plain text chat server, which can accept the connection of multiple clients at the same time, and forward the text sent by any client to all clients (including the sender).

### 1.2 Envs

MacOS13.6.3 (22G436), Apple Silicon.

IntelliJ IDEA2023.2.1. JDK19.

### 1.3 Quick Start

(1) run the `.jar` directly.

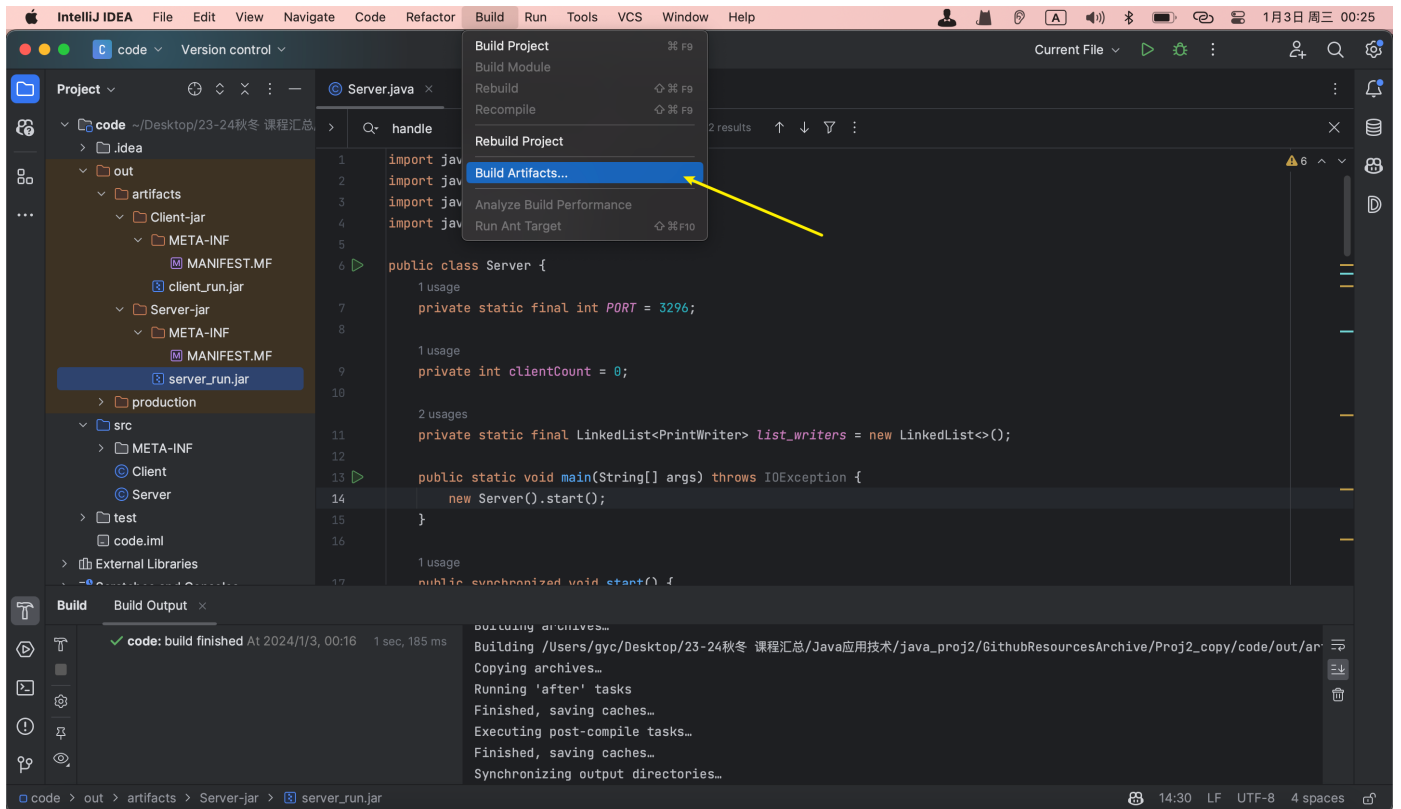
```
java -jar out/artifacts/Server-jar/server_run.jar
```

With server initialized, you can now run the `client_run.jar` repeatedly. Each run would generate a new client(auto numbered) connecting to the server via port 3296. Then you are free to test its functionalities using GUI popped up.

```
java -jar out/artifacts/Server-jar/client_run.jar
```

(2) compile from source code

Open the project using IntelliJ IDEA and click on "Build->Build artifacts...". You may need to set up JDK and adjust artifact parameters beforehand.



## 2 Code & Design

### 2.1 Server

```
6 public class Server {
7     private static final int PORT = 3296;
8
9     private int clientCount = 0;
10
11     private static final LinkedList<PrintWriter> list_writers = new LinkedList<>();
12
13     public static void main(String[] args) throws IOException {...}
14
15     public synchronized void start() {...}
16
17     private ServerSocket setupServerSocket() throws IOException {...}
18
19     private void handleNewClient(Socket clientSocket) {...}
20
21     public void broadcast(String msg) {...}
22
23     public class ServerClient implements Runnable {...}
24 }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
```

The `Server` class is a network server for handling multiple client connections and messaging, based on Java's socket programming. It is designed to listen for client connection requests on a specified port and create a new thread for each connected client to handle message communication.

The following is a description of its design and key methodology:

### 1. Macro-functionality:

- **Multi-Client Processing:** The server is capable of handling multiple client connections and message transfers. This is accomplished by creating a new thread for each connected client, thus supporting concurrent processing.
- **Message broadcasting:** The server receives messages from any client and broadcasts them to all connected clients.

### 2. Introduction to several key methods:

- **`start()`:** This is the main entry point for the server to run. It first sets up the server socket and enters an infinite loop that constantly listens for new client connection requests. Once it accepts a new connection, it calls the `handleNewClient()` method to handle that client.
- **`setupServerSocket()`:** This method is responsible for creating and setting up a `ServerSocket` on the specified port. This is the point at which the server receives a new connection.
- **`handleNewClient(Socket clientSocket)`:** This method is called whenever a new client connection is made. It creates a `ServerClient` object and starts a new thread for each new client.
- **`broadcast(String msg)`:** This method is used to broadcast the received message to all connected clients. This is accomplished by traversing the `list_writers` linked list storing all client output streams and sending the message.
- **`initializeStreams(int clientCount)`:**

```
private void initializeStreams(int clientCount) {
    try {
        in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()), true);
        out.println(clientCount);
        out.flush();
        list_writers.add(out);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

In the `initializeStreams` method, buffering is provided for input and output streams using the `BufferedReader` and `BufferedWriter` wrapper bases. This design reduces the need for frequent physical I/O operations and improves data processing efficiency by storing data in internal buffers. For input streams, this means that data can be read into memory in batches and then processed one at a time; for output streams, data is first accumulated in memory and then sent out at once, reducing the number of network communications.

In particular, `PrintWriter`, used in output streams, has auto-refresh enabled in its constructor (`true` parameter). This ensures that data is sent from the buffer immediately after each call to `println` or a similar method, rather than waiting with a delay. This is particularly important to ensure timely data transfer, especially in network communication scenarios such as sending a client's unique identifier to each connected client.

### 3. `ServerClient` class (important):

- An internal class that handles message communication for a single client. Each `ServerClient` instance runs in its own thread and is responsible for reading messages from its corresponding client and calling the `broadcast` method to broadcast them.
- In the `initializeStreams` method, input and output streams are set up for each connected client, and the output streams are added to the `list_writers` linked list for subsequent message broadcasts.

## 2.2 Client

```

12 public class Client {
13     public static void main(String[] args) throws IOException {
14         Client client = new Client();
15         client.initFrame(); // 初始化窗口GUI
16         client.initConnect(); // 初始化"S-C"连接
17     }
18
19     1 usage
20     private static final int PORT = 3296; // 和server端规定的端口号一致
21     8 usages
22     private Socket socket;
23     9 usages
24     private PrintWriter out;
25     5 usages
26     private BufferedReader in;
27     4 usages
28     private JTextArea textArea;
29     9 usages
30     private JFrame frame;
31     5 usages
32     private int id;
33
34     1 usage
35     void initFrame() {...}
36
37     1 usage
38     private JScrollPane createMessageDisplayArea() {...}
39
40     no usages
41     private JPanel createInputPanel() {...}
42
43     1 usage
44     private void sendMessage(String message) {...}
45
46     1 usage
47     private void sendExitMessage() {...}
48
49     1 usage
50     private JPanel getJPanel() {...}
51
52     1 usage
53     private void closeResources() {...}
54
55     1 usage
56     public void initConnect() throws IOException {...}
57
58     1 usage
59     private void setupNetworkConnection() throws IOException {...}
60
61     2 usages
62     public class MsgReceiver implements Runnable {...}
63 }

```

For BEST user experience, the client class involves constructing user-friendly graphical interface.

More specifically, the Client class is a Java Swing-based graphical user interface (GUI) application used to communicate with a socket-based server. It's capable of establishing a connection with the server, sending and receiving messages, and handling user interface interactions. It provides an input box for the user to enter messages through a graphical interface and the ability to display messages received from the server in a text area.

Key methods include `initFrame` for setting up the client's GUI, `initConnect` for initializing the connection to the server and starting a thread to receive messages, and `sendMessage` and `sendExitMessage` for sending messages and exit messages on close.

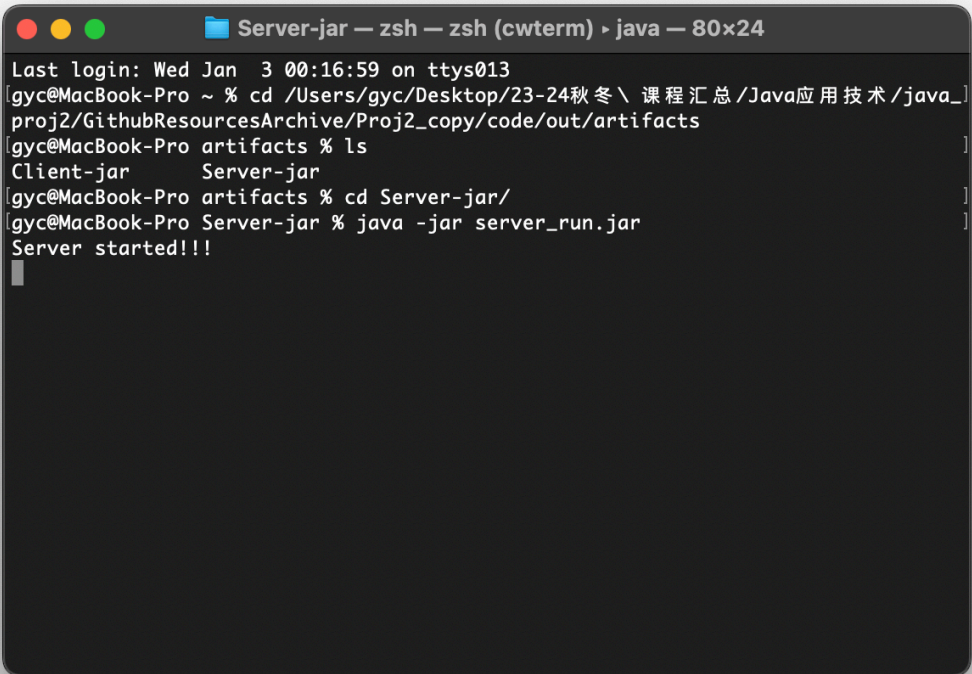
The `initConnect()` function requests the local server to establish a connection based on the PORT, and initializes the in/out pipeline.

These methods work together to ensure a responsive user interface and effective network communication, allowing the client to interact with the server and other clients in real time.

## 3 Test

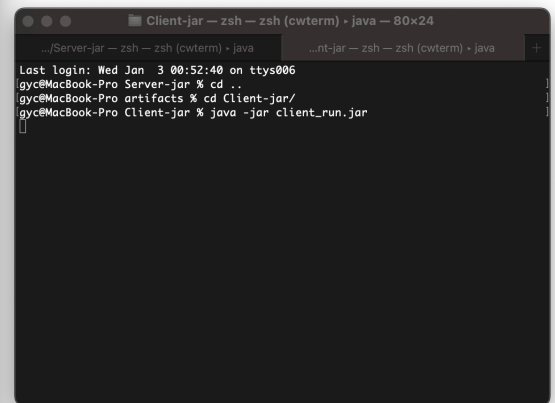
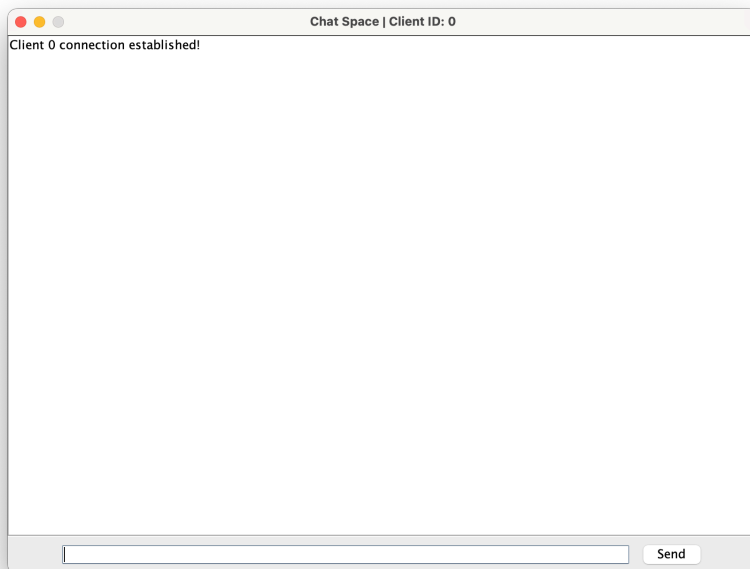
### 3.1 Normal test

Start server:

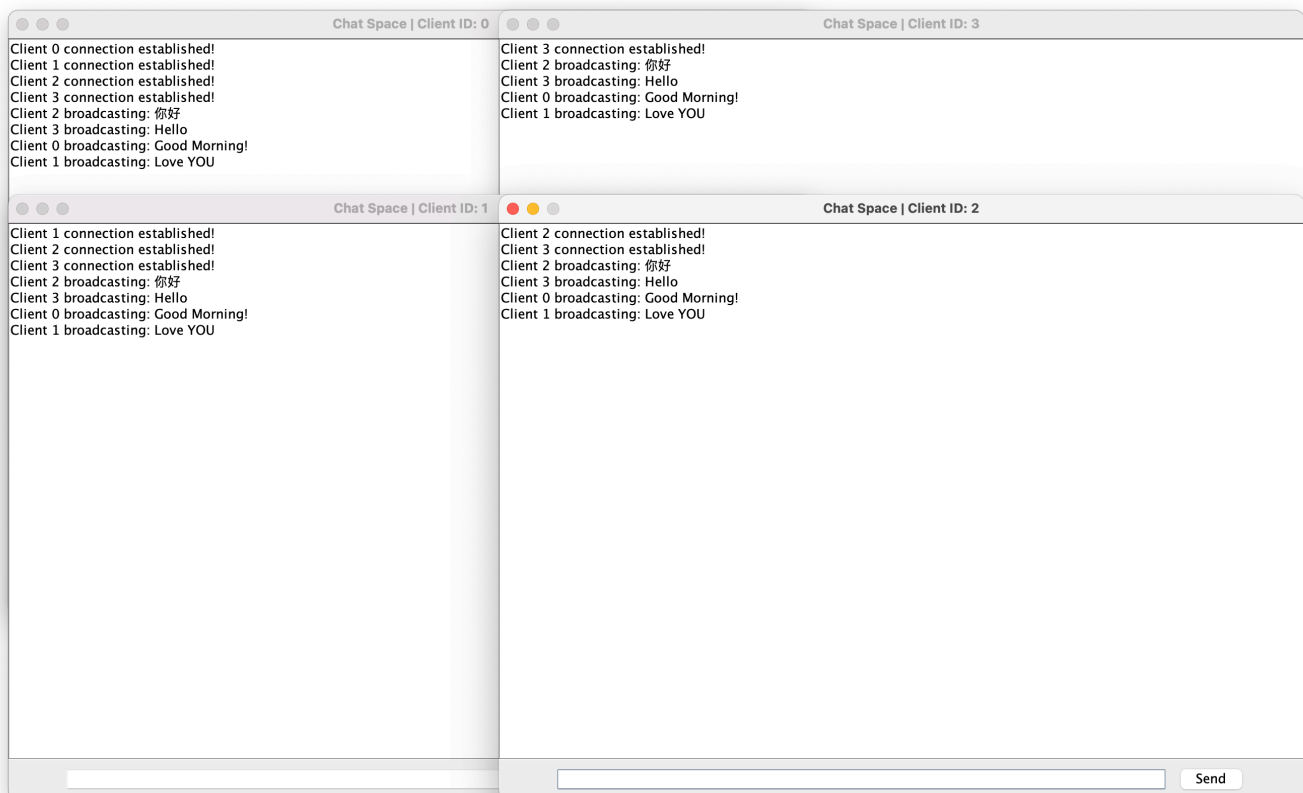


```
Server-jar — zsh — zsh (cwtterm) ▸ java — 80x24
Last login: Wed Jan  3 00:16:59 on ttys013
[gye@MacBook-Pro ~ % cd /Users/gye/Desktop/23-24秋冬\ 课程汇总/Java应用技术/java_
proj2/GithubResourcesArchive/Proj2_copy/code/out/artifacts
[gye@MacBook-Pro artifacts % ls
Client-jar      Server-jar
[gye@MacBook-Pro artifacts % cd Server-jar/
[gye@MacBook-Pro Server-jar % java -jar server_run.jar
Server started!!!
```

Create clients:



Multiple clients at separate windows, sending and broadcasting messages at the same time:



## 3.2 Stress test

I've also designed a stress test with python. It is expected to give us a rough idea on how much chat-forwarding pressure the server can handle.

```

1  import socket
2  import threading
3  import random
4  import string
5  import time
6
7  # 定义测试常量
8  NUM_MESSAGES = 15000
9  NUM_CLIENTS = 3
10 MESSAGE_MIN_LENGTH = 10
11 MESSAGE_MAX_LENGTH = 50
12 HOST = 'localhost'
13 PORT = 3296
14
15 def random_string(length):
16     """生成一个指定长度的随机字符串"""
17     letters = string.ascii_letters + string.digits
18     return ''.join(random.choice(letters) for i in range(length))
19
20 def client_task():
21     """客户端任务，发送指定数量的随机消息"""
22     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
23         sock.connect((HOST, PORT))
24         client_id = sock.recv(1024).decode()
25         print(f"Connected as Client {client_id}")
26
27         for _ in range(NUM_MESSAGES):
28             message_length = random.randint(MESSAGE_MIN_LENGTH, MESSAGE_MAX_LENGTH)
29             message = random_string(message_length)
30             sock.sendall(f"Client {client_id} broadcasting: {message}".encode())
31
32         print(f"Client {client_id} finished sending messages")
33
34 # 记录开始时间
35 start_time = time.time()
36
37 # 创建并启动指定数量的客户端线程
38 threads = []
39 for _ in range(NUM_CLIENTS):
40     thread = threading.Thread(target=client_task)
41     thread.start()
42     threads.append(thread)
43
44 # 等待所有线程完成
45 for thread in threads:
46     thread.join()
47
48 # 记录结束时间并计算总耗时
49 end_time = time.time()
50 total_time = end_time - start_time
51 print(f"All messages sent in {total_time:.2f} seconds.")
52

```

You can edit the testing parameters according to your needs.

# 定义测试常量

NUM\_MESSAGES = 15000

NUM\_CLIENTS = 3

MESSAGE\_MIN\_LENGTH = 10

MESSAGE\_MAX\_LENGTH = 50

HOST = 'localhost'

PORT = 3296

To conduct the test, simply use the command below (assuming you have all the modules needed installed)



```
python test/StressTest.py
```

I recorded series of test datas in tables and drew them in graphs.

	A	B	C	D	E	F
1	NUM_MESSAGES	2000	6000	10000	12000	15000
2	NUM_CLIENTS = 3	0.19	0.34	0.96	1.23	5.24
3	NUM_CLIENTS = 4	0.21	0.36	0.89	1.12	7.21
4	NUM_CLIENTS = 5	0.24	0.35	1.01	1.47	8.54

