Design Document for P2P Project

jiahe.zhang 20205722

November 2022

1 Design requirements

1.1 Mastering the principles of P2P

1.2 Implementation of an intra-LAN messaging system

1.3 Function

Create a P2P messaging system on a LAN, where the program is both a server and a client, with its own port number for the server.

1.3.1 User registration and access to peer lists

When peer A starts up, the user sets up his own information (username, group); scans the network segment for online peers (the server port is open) and sends a message to the service ports of all online peers; the receiver receives the message, adds peer A to its own user list and sends an answer message; peer A adds the other peers who respond to the message to its user list. The format of the messages exchanged between the two parties is defined according to their needs and includes at least the user name and IP address.

1.3.2 Sending messages

The user selects the user in the list, establishes a TCP connection with the user and sends a message. The sending of messages has specific structural requirements and different message structures correspond to different functional implementations.

1.4 User Interface

The interface includes a list of peers; a message display list; a message input box; a file transfer process display and action buttons or menus.

2 Tools & Environment

Language: Java Environment: IDEA JDK: 1.8 System: Windows 10 Operating Requirement: Operating system independent, Windows, Linux, Mac OS X can run with the Java runtime environment JRE installed.

3 Basic knowledge of program development

3.1 The basic principles and communication mechanisms of Socket and TCP

3.1.1 TCP connections

Three handshakes are required to establish a TCP connection. First handshake: the client sends a syn packet (syn=j) to the server and enters the SYN_SEND state, waiting for the server to acknowledge it. Second handshake: the server receives a syn packet and must acknowledge the client's SYN (ack=j+1) and at the same time send a SYN packet (syn=k) itself, i.e. a SYN+ACK packet, at which point the server enters the SYN_RECV state. The third handshake: the client receives a SYN + ACK packet from the server and sends an acknowledgement packet ACK (ack=k+1) to the server, this packet is sent and the client and server enter the ESTABLISHED state, completing the three handshakes. The packet transmitted during the handshake does not contain data, and only after the three handshakes are complete does the client and server formally begin transmitting data. Ideally, once established, a TCP connection is maintained until either party actively closes the connection. When disconnecting, both the server and the client can initiate a request to disconnect the TCP connection, which requires a "four handshake" process.

3.1.2 SOCKET Principle

The socket is the cornerstone of communication and is the basic unit of operation for network communication supporting the TCP/IP protocol. It is an abstract representation of the endpoints in the network communication process and contains the five types of information necessary for network communication: the protocol used for the connection, the IP address of the local host, the protocol port of the local process, the IP address of the remote host and the protocol port of the remote process. When the application layer communicates data through the transport layer, TCP encounters the problem of providing concurrent services to multiple application processes at the same time. Multiple TCP connections or multiple application processes may need to transfer data over the same TCP protocol port. To distinguish between different application processes and connections, many computer operating systems provide a socket interface for applications to interact with the TCP/IP protocol. The application layer and transport layer can use the socket interface to differentiate communications from different application processes or network connections, enabling concurrent services for data transfer.

3.2 Functional design and interface design

4 Overall design

4.1 Design thinking

4.1.1 The classic server-client architecture for TCP communications

The server has a server-waiting-for-user-connection thread that loops through the client's TCP connection requests. Once a connection request has been captured with ServerSocket.accept(), a client service thread is assigned to that TCP connection and the server communicates with the client through this messaging thread. The server sends messages through the methods of this client service thread in the main thread, while the incoming messages are all received and processed in a loop in the client service thread. The client can initiate a socket connection request to the server, and once it receives a successful response from the server to the connection request, the client assigns a message receiving thread to the socket, otherwise the socket is closed. similar to the server task assignment, sending messages is done in the main thread as a non-useful method, while receiving messages are constantly refreshed and processed accordingly in the message receiving thread.

4.1.2 Unified ASCII level text transfer protocol

Server information

1. Chat messages

MSG @	TO @	FROM @	CONTENT_TYPE @	CONTENT
	ALL	SERVER		xxx
	ALL	USER	BROADCAST	xxx
	USER	USER	MESSAGE	xxx

2. Login status

LOGIN @	status @	content
	SUCCESS	xxx
	FAIL	xxx

3. User operations

USER @	type @	other
	ADD	USER
	DELETE	USER
	LIST	number @ USER
	KICK@	USER
	STATS@	USER

4. Error messages

ERROR @	TYPE
	xxx

5. Server shutdown

CLOSE

Client Message Format

1. Chat messages

MSG @	TO @	FROM @	CONTENT_TYPE @	CONTENT
	ALL	USER	BROADCAST	xxx
	USER	USER	MESSAGE	xxx

2. Login notification

LOGIN @	USER

3. Logout notice

STOP

4.1.3 MVC layered model

Model-View-Controller is a classic application development design pattern, it is about data management, interface display and user interaction, program maintenance management are encapsulated in the MVC three categories, enough to become loosely coupled relationship. This course design I also use the MVC design ideas, independent Model class User for saving client user information, DefaultListModel model class for storing online user queue; View will be placed in a separate package, Controller listen to user operation events, reflect to the Model class processing and update in the View. The idea of MVC is that there should be no direct connection between M and V. The business logic is encapsulated in MC, while V is only responsible for display. This experiment binds the respective Listener for the V class to listen to user operations, completes the business logic processing in C, saves and updates the User and DefaultListModel, and finally displays it on the UI interface.

4.1.4 concurrentHashMap manages thread queues and user lists

concurrentHashMap is a multi-thread-safe hash table defined in the java.util.concurrent package. The use of hash tables to manage thread queues and user lists can be quickly indexed, and multi-thread safety also ensures data consistency for shared resources between multiple user service threads.

4.2 Procedure flow chart

4.3 Key data structures

4.3.1 User

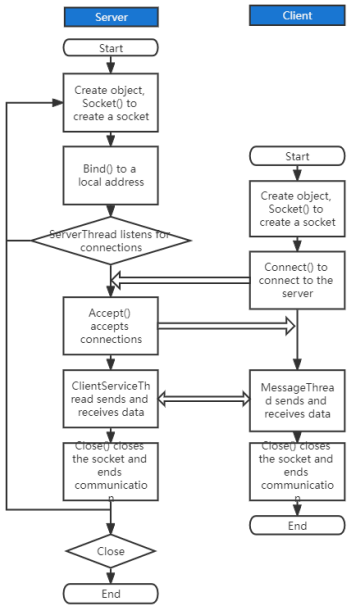

Figure 1: Procedure flow chart

Constructed-There are two, one instantiates a User with a separate name and IP, and the other instantiates a User with a string of name%IP spliced together

Read-only-name and ipAddr are both private, give them a read-only getter

description()-Returns a string of name%IP stitched together to represent an individual user

4.3.2 ServerView

UI-The initUI() constructor mostly sets up the UI interface, which uses GridLayout and BorderLayout. serviceUISetting(false) is used to turn off all the buttons and textFields that only work when they are connected (false is changed to true to turn them on, and all setting-related buttons and textFields).

4.3.3 Server

1. startServer() - startButton binding First check the maxClientNum and port for legal input, if not pop up an error window and exit. Then initialize the concurrent hash table clientServiceThreads that manages the queue of client service threads, initialize the server-Socket that listens for client connection requests, and initialize and open a thread that listens for connection requests. Finally there is some error handling and server logging.

2. request listener thread ServerThread isRunning is used as a thread run flag bit to control the survival of the thread, and the listening logic is done in the function run() which is called after the thread starts. If it is on, it keeps looping and serverSocket.accept() is blocking and the thread will not run until another thread/process requests a socket connection from it. This is the reason for one of the bugs I mention below: accept() blocks the thread it keeps waiting, and just ending the thread with a flag bit doesn't get it out of the blocking state (it hasn't looped to the next while judgement), so I force close the serverSocket in closeThread() and an exception is thrown! After receiving the connection request accept() returns a socket which is used to communicate with the client requesting the connection. At this point the 3 handshakes to establish a TCP connection have been completed, all encapsulated by the serverSocket class. After the communication socket is obtained, the server is checked to see if the number of people online is full and a login success or failure message is sent to the client. If the connection is not full, a clientServiceThread thread is assigned to the client specifically for sending and receiving TCP packets from the client.

3. listening for client messages ClientServiceThread Key fields-The reader is used to read the client's message input, and the readLine method also blocks until the client has sent a message. The writer has a write buffer and a flush() function that forces the message to be sent and refreshes the buffer, I encapsulate the write message in sendMessage(String).

Initialization-The initialization starts by binding the reader and writer to the socket response stream, before determining whether the message sent in the user's socket request is in the correct format (the thread will not execute if it is not). The new user is then notified to all users who are already online, sending the notification requires traversing the entire service thread queue and sending the various notifications defined in the text transfer protocol. Notice that the service thread is not added to the service thread queue at this point, it is added after initialisation is complete. After notifying the other users that the new client is online, the client is then told which users are now online, again using the protocol format for sending notifications. Here the StringBuffer class is used, which is more efficient than String's + when multiple strings are connected.

Thread running-If it is a LOGOUT message informing the client that it is offline, it sends a message to all other client processes that the user is offline and deletes the user object from the model class and the thread from the thread queue. If it is a message, it is handed over to the dispatchMessage(String) method to distribute the message specifically.

DispatchMessage()-This method parses the to field of the MSG class message and selects whether to send the message to a specific user or to send it directly to a group of users based on the to field. If the message is sent to a specific user, it is indexed by the to field (userDescription) to quickly find the thread serving the user's client from the service thread queue to send the message.

Description-Although the ServerMain class sends messages through a writer in the ClientServiceThread and is also called from within this Thread inner class, the call to the writer to sendMessage is not necessarily done within the thread (the thread refers to the inside of the while loop in run()). The main job of the ClientServiceThread is to listen to the messages sent to the server by each client.

4.3.4 ClientView

UI-The initUI() constructor mostly sets up the UI interface, which uses GridLayout and BorderLayout. serviceUISetting(false) is used to turn off all the buttons and textFields that only work when they are connected (false is changed to true to turn them on, and all setting-related buttons and textFields).

4.3.5 Client

connect to the server-Error detection here does not determine the legitimacy of the IP address, and the determination is not very troublesome. When the user input is legitimate, a socket is instantiated based on the server IP address and port, which is used for future communication with the server. The client's local IP address is obtained and instantiated with this IP address, and a message with its own user information (name and IP) is sent to the server via the socket to indicate the request. Immediately after sending, the MessageThread is opened to wait for a response from the server.

MessageThread receives messages from the server-reader.readLine() blocks the reading of the server message. The StringTokenizer class passes a String and a split string instance or a tokenizer, which gets a series of tokens by splitting the string and passing it through tokenizer.nextToken() to get the token string. It is easy to see that it does the same thing as String.split(String), but it is significantly more efficient than the split method (a more detailed performance analysis can be found in a Baidu search comparing the two).

5 Tests & Benefits

1. Run Server and Client-This program supports a single server with multiple clients. Both the server and the client cannot send messages but modify the configuration in the unenabled/connected state, and cannot modify the configuration but send messages in the enabled/connected state, and the UI logic is determined by the enabled/connected state. The server configuration can modify the listening port and the maximum number of listeners, and when it is on, it receives all messages from the client and forwards them by mimicking the packet parsing action.

Clients can set their own name, server IP and server port. The client can select everyone in the online user list or other online users to send messages to after they have connected. Information about other users going online and offline is indicated in the online user list and in the chat message box.

In a P2P network, nodes can be added and removed at will. New nodes are added to the network, usually through a seed node that accesses the network, while the seed node broadcasts the new node to other nodes and makes connections.

In a P2P network, there is no strict distinction between client and server, and each node acts as both a client and a server. The nodes are equal to each other and any node is able to inform every node in the network of the message as long as it is connected to the network.

The messaging uses a tokenizer for field splitting, which has been tested to be significantly more efficient and faster than the split case.

The fundamental non-centralised nature of P2P brings with it advantages in terms of scalability and robustness. In a P2P network, not only does the demand for services increase as users join, but the overall resources and service capabilities of the system expand in parallel, always making it easier to meet the needs of users.

6 Weaknesses & Improvements

When the server was closed, I shut down the thread where the server received the socket request and closed the ServerSocket, but the thread still continued to execute ServerSocket.accept() once. I tried to use the synchronized method and determine if the ServerSocket was closed, but the exception still occurs. I caught the exception and just printStackTrace without doing any other error handling, it did not affect the server shutdown and all clients received the server shutdown message correctly.

As each node performs message forwarding, it results in the same node receiving the same message multiple times, which leads to duplication of messages and greater redundancy.

When there are too many clients, the efficiency of the service drops significantly.

The data structure has the potential to waste storage space.

The division of transmitted information can be simplified and categorised.