컴퓨터네트워크

Design Project

남윤찬 (20223069, 소프트웨어학부)
신진욱 (20223098, 소프트웨어학부)
조다운 (20223139, 소프트웨어학부)
황현진 (20223158, 소프트웨어학부)

• 개요

이 메신저 프로그램은 클라이언트-서버 아키텍처로 구현된 복수 사용자 메신저 프로그램입니다. 주요 요구 사항을 충족하며 텍스트 기반의 간단한 UI를 제공하여 사용자들이 메시지를 주고받고 세션을 관리할 수 있게 합니다.
해당 시스템은 세 개의 주요 클래스 (ChatClient, ChatServer, ChatThread)로 구성되어 있으며, 네트워크 통신과 사용자 입력 처리에 중점을 두어 설계되었습니다.
또한 이 프로그램은 확장성을 고려하여 설계 되었으며, 헤더와 바디 부분으로 구분된 메시지 포맷을 통해 향후 추가 기능을 쉽게 구현할 수 있습니다.

• 과제 요구사항

요구사항		구현 여부
메신저 프로그램은 로그인 서버와 사용자 프로그램으로 구성된다. 로그인 서버는 현재 online인 사용자의 목록을 저장하고 있다. 사용자의 목록에는 사용자 별 아이디, IP 주소, Port 번호를 저장한다. (저장은 파일에 하면 된다.)		O
새로 클라이언트(사용자 프로그램)가 실행되면 로그인 서버에 자신의 아이디와 IP 주소, 포트 번호를 알려주고, 로그인 서버가 저장하고 있는 다른 online 사용자 목록을 받는다. 이 사용자 목록을 화면에 출력하는 방식으로 해서 사용자에게 어느 아이디를 가진 사용자가 현재 online인지 알 수 있게 한다.		O
클라이언트는 로그인 서버로부터 받은 목록을 활용하여 다른 사용자에게 접속을 하는데, 서로 간의 메시지는 로그인 서버를 거치지 않고, 직접 전송하게 된다.		O
사용자들은 서로 간의 아이디를 이용하여 메시지를 주고 받을 수 있도록 UI를 만든다.		O
사용자는 프로그램을 실행한 후 다음과 같은 기능을 수행할 수 있다.	사용자를 메신저 세션에 초청한다.	O
	메신저 세션을 끝낸다.	O
	사용자가 해당 메신저 세션에 있는 모든 사용자에게 메시지를 보낸다.	O
설계 시 고려 사항	시간: 설계를 단순하게 하여 0.5 man-month 정도의 분량으로 한다.	O
	사용 편의성: UI는 텍스트 기반 UI로 만드는데, 최대한 단순하게 해도 된다. 위의 3가지 기능만 실행되면 큰 문제 없다.	O
	메시지 포맷: 메시지 포맷은 HTTP Request나 HTTP Response 메시지와 비슷하게 헤더 부분과 바디 부분으로 구분하게 하여, 추후 헤더 필드를 추가하여 부가적인 기능을 추가할 수 있는 확장성을 가지도록 한다.	O

• 클래스 상세 설명

< ChatClient 클래스 >

이 클래스는 클라이언트 프로그램을 구현하며 다음과 같은 기능을 합니다.

1. **클라이언트 초기화:**
   - 사용자가 프로그램을 실행할 때 닉네임을 인수로 받습니다. args.length != 1 조건을 확인하여 잘못된 실행을 방지합니다.
   - 서버에 소켓을 연결하고 사용자 입력을 받을 준비를 합니다.
   - 닉네임과 함께 클라이언트의 IP와 포트정보를 서버로 전송합니다.

```java
package chat;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

public class ChatClient {
    Run | Debug
    public static void main(String[] args) throws Exception {
        if(args.length != 1){
            System.out.println(x:"command: java .\\ChatClient 닉네임");
            return;
        }
        //사용자 이름
        String name = args[0];
        //server에 socket 연결
        Socket socket = new Socket(host:"127.0.0.1", port:8888);
        //사용자 입력 받기 및 전송
        BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter pw = new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
        BufferedReader keyboard = new BufferedReader(new InputStreamReader(System.in));

        // 닉네임 전송
        pw.println(name);
        pw.flush();
```

2. **메시지 송수신:**

- InputThread를 생성하여 서버로부터 오는 메시지를 백그라운드에서 받습니다.
- 키보드 입력을 받아서 서버로 메시지를 전송합니다.

```
28          // 백그라운드로 서버가 보내준 메세지를 읽여들여서 화면에 출력
29          InputThread inputThread = new InputThread(br);
30          inputThread.start();
31
32          // 서버에 메시지, 명령어 전송
33          try{
34              String line = null;
35              while((line = keyboard.readLine()) != null) {
36                  if("/quit".equals(line)) {
37                      pw.println(x:"/quit");
38                      pw.flush();
39                      break;
40                  }
41                  pw.println(line);
42                  pw.flush();
43              }
44          }catch(Exception ex){
45              System.out.println(x:"...");
46          }
47
48          br.close();
49          pw.close();
50
51          //연결 종료
52          System.out.println(x:"socket close!!");
53          socket.close();
54
55      }
56  }
```

+) InputThread:

- 클라이언트가 서버로부터 오는 메시지를 비동기적으로 처리할 수 있게 합니다.
- run() 메서드는 서버로부터 메시지를 읽어들여서 콘솔에 출력합니다.

```
58  //서버로부터 메시지 받기
59  class InputThread extends Thread {
60      BufferedReader br;
61      public InputThread(BufferedReader br){
62          this.br = br;
63      }
64
65      @Override
66      public void run() {
67          try{
68              String line = null;
69              while((line = br.readLine()) != null){
70                  System.out.println(line);
71              }
72          }catch(Exception ex){
73              System.out.println(x:"...");
74          }
75      }
76  }
```

< ChatServer 클래스 >

이 클래스는 서버 프로그램을 구현하며 다음과 같은 기능을 합니다:

1. 서버 초기화:

- 포트 8888에서 ServerSocket을 열고 클라이언트 연결 요청을 기다립니다.
- 연결이 수립되면 각 연결에 대해 ChatThread를 생성하여 클라이언트와의 통신을 개별적으로 관리합니다.

2. 온라인 사용자 관리:

- 모든 온라인 사용자 목록을 관리하고, 각 사용자의 세션 번호를 부여합니다.

```java
package chat;

import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ChatServer {
    Run | Debug
    public static void main(String[] args) throws Exception {
        ServerSocket serverSocket = new ServerSocket(port:8888);

        //thread로 된 유저들의 리스트(온라인 유저 목록)
        List<ChatThread> onlineList = Collections.synchronizedList(new ArrayList<>());
        //session 번호: 기본값으로 defaultSession++ 값들이 들어가 자동적으로 같은 세션에 들어가지 않게 한다
        int defaultSession = 0;

        while(true) {
            Socket socket = serverSocket.accept();
            ChatThread chatThread = new ChatThread(socket, onlineList, defaultSession++);
            chatThread.start();
        }
    }
}
```

< ChatThread 클래스 >

이 클래스는 각 클라이언트와의 통신을 처리하며, 주요 기능은 다음과 같습니다:

1. 클라이언트 초기화:

- 소켓을 통해 클라이언트와 통신할 BufferedReader와 PrintWriter를 초기화합니다.
- 클라이언트로부터 닉네임을 받아서 온라인 사용자 목록에 추가합니다.

```java
package chat;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class ChatThread extends Thread {

    private String name;
    private BufferedReader br;
    private PrintWriter pw;
    private Socket socket;
    List<ChatThread> onlineList;
    private int sessionNum;

    public ChatThread(Socket socket, List<ChatThread> onlineList, int defaultSession)throws Exception {

        this.socket = socket;
        BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintWriter pw =  new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
        this.br = br;
        this.pw = pw;
        this.name = br.readLine();
        this.onlineList = onlineList;
        this.onlineList.add(this);
        this.sessionNum = defaultSession;

    }
```

2. 메시지 브로드캐스트:

- sendMessage(String msg) 메서드를 통해 특정 클라이언트로 메시지를 보냅니다.

```
38      //메시지 전송
39      public void sendMessage(String msg){
40          pw.println(msg);
41          pw.flush();
42      }
```

- broadcast(String msg, boolean includeMe) 메서드를 통해 같은 세션에 있는 모든 사용자에게 메시지를 전송합니다.

```
123     private void broadcast(String msg, boolean includeMe){
124         List<ChatThread> chatThreads = new ArrayList<>();
125         for(int i=0;i<this.onlineList.size();i++){
126             chatThreads.add(onlineList.get(i));
127         }
128
129         try{
130             for(int i=0;i<chatThreads.size();i++){
131                 ChatThread ct = chatThreads.get(i);
132                 if(!includeMe){ //메시지 작성자를 포함하지 않음
133                     if(ct == this){
134                         continue;
135                     }
136                 }
137                 if(ct.sessionNum != this.sessionNum) {
138                     continue;
139                 }
140                 ct.sendMessage(msg);
141             }
142         }catch(Exception ex) {
143             System.out.println(x:"///");
144         }
145     }
```

3. 사용자 목록 업데이트:

- updateUserList() 메서드를 통해 현재 온라인 사용자 목록을 userList.txt 파일에 저장합니다.

```
44      //userlist 업데이트
45      public void updateUserList() {
46          try {
47              File file = new File(pathname:"./userList.txt");
48
49              if(!file.exists()) {
50                  file.createNewFile();
51              }
52
53              FileWriter fw = new FileWriter(file);
54              BufferedWriter fileWriter = new BufferedWriter(fw);
55
56              fileWriter.write(this.onlineList.toString());
57              fileWriter.close();
58
59          } catch (IOException e) {
60              e.printStackTrace();
61          }
62
63
64      }
```

- userList.txt (user1이 /quit하고 퇴장한 후, user2만 남은 상황)

```
userList.txt
1  [user2 /127.0.0.1 49249]
```

4. 세션 관리:

- getOnlineList(boolean includeMe) 메서드는 온라인 사용자 목록을 특정 클라이언트에게 전송합니다.

```
147     private void getOnlineList(boolean includeMe) {
148         List<ChatThread> chatThreads = new ArrayList<>();
149         for(int i=0;i<this.onlineList.size();i++){
150             chatThreads.add(onlineList.get(i));
151         }
152 
153         try{
154             for(int i=0;i<chatThreads.size();i++){
155                 if(chatThreads.get(i) == this){
156                     chatThreads.get(i).sendMessage(chatThreads.toString());
157                     break;
158                 }
159             }
160         }catch(Exception ex) {
161             System.out.println(x:"///");
162         }
163     }
```

- getSessionMembers(int sessionNum) 메서드는 특정 세션에 있는 사용자 목록을 전송합니다.

```
165     private void getSessionMembers(int sessionNum) {
166         List<ChatThread> chatThreads = new ArrayList<>();
167         for(int i=0;i<this.onlineList.size();i++){
168             if(this.sessionNum == this.onlineList.get(i).sessionNum){
169                 chatThreads.add(this.onlineList.get(i));
170             }
171         }
172 
173         for(int i=0;i<chatThreads.size();i++){
174             if(chatThreads.get(i) == this){
175                 chatThreads.get(i).sendMessage(chatThreads.toString());
176                 break;
177             }
178         }
179     }
```

- invite(String name) 메서드는 특정 사용자를 현재 세션으로 초대합니다.

```
186     private void invite(String name) {
187         for(int i=0;i<this.onlineList.size();i++){
188             if(name.equals(this.onlineList.get(i).name)){
189                 this.onlineList.get(i).sessionNum = this.sessionNum;
190                 broadcast(this.name + "님의 session에 " + name + "님이 연결되었습니다.", includeMe:true);
191             }
192         }
193     }
```

5. 메시지 전송 및 명령어 처리, 클라이언트 종료:

- 클라이언트가 보낸 메시지를 처리합니다.
- 세션 관리 명령어(/onlineList, /sessionList, /invite) 처리를 하며, “{닉네임}님의 연결이 끊어졌습
니다.”와 같은 메시지를 모든 사용자에게 전송합니다.
- /quit 명령어를 처리하여 클라이언트가 연결을 종료할 수 있게 합니다.

```
66      @Override
67      public void run() {
68          updateUserList();
69
70          //ChatThread는 사용자가 보낸 메세지를 읽어들임
71          try{
72              broadcast(name + "님이 연결되었습니다.", includeMe:false);
73              String line = null;
74              while((line = br.readLine()) != null){
75                  //세션 이탈
76                  if("/quit".equals(line)){
77                      break;
78                  }
79                  //온라인 유저 목록 확인
80                  if(line.startsWith(prefix:"/onlineList")) {
81                      getOnlineList(includeMe:true);
82                      continue;
83                  }
84                  //본인의 세션에 참가한 유저 확인
85                  if(line.startsWith(prefix:"/sessionList")) {
86                      getSessionMembers(this.sessionNum);
87                      continue;
88                  }
89                  //name을 가진 유저 초대
90                  if(line.startsWith(prefix:"/invite")) {
91                      invite(line.split(regex:" ")[1]);
92                      continue;
93                  }
94                  //같은 세션의 유저들에게 메시지 전송
95                  broadcast(name + " : " + line,includeMe:true);
96              }
97          }catch(Exception ex){
98              //ChatThread가 연결이 끊어짐
99              ex.printStackTrace();
100         }
101         finally{
102             broadcast(name + "님이 연결이 끊어졌습니다.", includeMe:false);
103             this.onlineList.remove(this);
104             updateUserList();
105             try{
106                 br.close();
107             }catch(Exception ex){
108             }
109
110             try{
111                 pw.close();
112             }catch(Exception ex){
113             }
114
115             try{
116                 socket.close();
117             }catch(Exception ex){
118             }
119         }
120
121     }
```

• 작동 흐름

1. **클라이언트 연결 및 초기화**: 사용자는 클라이언트 프로그램을 실행하고 서버에 연결합니다. 연결이 성립되면, 사용자의 닉네임 및 네트워크 정보가 서버에 등록됩니다.

2. **메시지 송수신**: 사용자는 키보드로 입력한 메시지를 서버로 보내고, 서버는 이를 같은 세션의 다른 클라이언트에게 브로드캐스트합니다.

3. **세션 관리**: 사용자는 /invite, /sessionList, /onlineList 등의 세션 관리 명령어를 사용하여 다른 사용자를 초대하거나 현재 세션의 사용자 목록을 확인할 수 있습니다.

4. **종료**: 사용자가 /quit 명령어를 입력하면 클라이언트는 서버와의 연결을 종료하고, 서버는 해당 클라이언트를 온라인 사용자 목록에서 제거합니다.

• 사용 예시

1. 클라이언트 실행: java .\ChatClient 닉네임

2. 메시지 전송:
   - 일반 메시지: 입력한 텍스트는 같은 세션의 다른 클라이언트에게 전송됩니다.
   - 명령어: /onlineList, /sessionList, /invite 닉네임, /quit
     - **/onlineList** : 로그인 서버가 저장하고 있는 다른 online 사용자 목록을 받아 출력
     - **/sessionList** : 현재 참여중인 메신저 세션의 사용자 목록을 출력
     - **/invite 닉네임** : 다른 사용자를 현재 세션에 초대
     - **/quit** : 프로그램을 종료