Урок 7

Написание сетевого чата. Часть I

Написание клиентской и серверной части чата. Многопоточная обработка клиентских подключений.

[Написание серверной части](#)

[Написание клиентской части](#)

[Практическое задание](#)

[Дополнительные материалы](#)

Написание серверной части

**При рассмотрении серверной и клиентской части не будут рассматриваться моменты, описанные в методичке 6.*

Серверная часть состоит из:

- ServerApp — основной класс, содержащий метод main() и запускающий сервер.
- MyServer — класс, представляющий собой сервер.
- ClientHandler — класс, отвечающий за обмен сообщениями между клиентами и сервером.
- AuthService — интерфейс, описывающий сервис аутентификации на стороне сервера.
- BaseAuthService — класс, реализующий аутентификацию клиента через обычный список клиентов.

Класс ServerApp всего лишь запускает сервер.

```
public class ServerApp {
    public static void main(String[] args) {
        new MyServer();
    }
}
```

Всё что связано с работой сервера вынесено в отдельный класс MyServer. MyServer хранит список подключенных клиентов, предназначенный для управления соединением с клиентом и рассылкой сообщений. При подключении и авторизации клиент добавляется в этот список (через метод subscribe()), при отключении — удаляется (через unsubscribe()). Для блокировки возможности авторизоваться нескольким клиентам под одной учётной записью используется метод isNickBusy(), проверяющий занятость ника в текущем сеансе чата.

```
public class MyServer {
    private final int PORT = 8189;

    private List<ClientHandler> clients;
    private AuthService authService;

    public AuthService getAuthService() {
        return authService;
    }

    public MyServer() {
        try (ServerSocket server = new ServerSocket(PORT)) {
            authService = new BaseAuthService();
            authService.start();
            clients = new ArrayList<>();
            while (true) {
                System.out.println("Сервер ожидает подключения");
                Socket socket = server.accept();
                System.out.println("Клиент подключился");
                new ClientHandler(this, socket);
            }
        } catch (IOException e) {
            System.out.println("Ошибка в работе сервера");
```

```
        } finally {
            if (authService != null) {
                authService.stop();
            }
        }
    }

    public synchronized boolean isNickBusy(String nick) {
        for (ClientHandler o : clients) {
            if (o.getName().equals(nick)) {
                return true;
            }
        }
        return false;
    }

    public synchronized void broadcastMsg(String msg) {
        for (ClientHandler o : clients) {
            o.sendMsg(msg);
        }
    }

    public synchronized void unsubscribe(ClientHandler o) {
        clients.remove(o);
    }

    public synchronized void subscribe(ClientHandler o) {
        clients.add(o);
    }
}
```

Интерфейс AuthService описывает правила работы с сервисом авторизации: start() для его запуска; getNickByLoginPass() для получения ника по логину/паролю либо null, если такой пары логин/пароль нет; stop() для остановки сервиса. Простейшая реализация этого интерфейса BaseAuthService основана на использовании списка записей логин-пароль-ник, при запуске и остановке ничего не происходит, а поиск осуществляется перебором списка записей. Сервис авторизации в дальнейшем может быть доработан для использования с базой данных.

```
public interface AuthService {
    void start();
    String getNickByLoginPass(String login, String pass);
    void stop();
}

public class BaseAuthService implements AuthService {
    private class Entry {
        private String login;
        private String pass;
        private String nick;

        public Entry(String login, String pass, String nick) {
```

```
            this.login = login;
            this.pass = pass;
            this.nick = nick;
        }
    }

    private List<Entry> entries;

    @Override
    public void start() {
        System.out.println("Сервис аутентификации запущен");
    }

    @Override
    public void stop() {
        System.out.println("Сервис аутентификации остановлен");
    }


    public BaseAuthService() {
        entries = new ArrayList<>();
        entries.add(new Entry("login1", "pass1", "nick1"));
        entries.add(new Entry("login2", "pass2", "nick2"));
        entries.add(new Entry("login3", "pass3", "nick3"));
    }

    @Override
    public String getNickByLoginPass(String login, String pass) {
        for (Entry o : entries) {
            if (o.login.equals(login) && o.pass.equals(pass)) return o.nick;
        }
        return null;
    }
}
```

За общение сервера с каждым отдельным клиентом отвечает обработчик - класс ClientHandler. Как только клиент подключается, создается новый объект ClientHandler, который будет аутентифицировать клиента и получать от него сообщения.

При старте обработчика клиента запускается отдельный поток, читающий все сообщения от клиента. В этом потоке первым делом попадаем в цикл аутентификации (метод authentication()): сервер ожидает от клиента сообщения вида «/auth login password», при получении разбивает его на части и проверяет наличие учётной записи с таким логином/паролем, если запись есть и не занята другим пользователем, отсылаем клиенту сообщение об успешной авторизации и его ник (например, «/authok nick1») рассылаем всем клиентам сообщение о том, что подключился новый участник, подписываем этого участника на рассылку чата и выходим из цикла авторизации. Если авторизация по какой-то причине не удалась, отсылаем клиенту сообщение с причиной отказа.

После выхода из цикла авторизации попадаем в обычный цикл обмена сообщениями до тех пор, пока клиент не пришлёт команду «/end», в результате которой выкидываем его из списка рассылки, закрываем сокет и завершаем поток чтения сообщений от него. Также каждый ClientHandler получил

ссылку на сервер, к которому он прикреплён для возможности обратиться к методам этого сервера. Поле name отвечает за ник клиента, если name пуст, клиент считается неавторизованным.

```
public class ClientHandler {
   private MyServer myServer;
   private Socket socket;
   private DataInputStream in;
   private DataOutputStream out;

   private String name;

   public String getName() {
       return name;
   }

   public ClientHandler(MyServer myServer, Socket socket) {
       try {
           this.myServer = myServer;
           this.socket = socket;
           this.in = new DataInputStream(socket.getInputStream());
           this.out = new DataOutputStream(socket.getOutputStream());
           this.name = "";
           new Thread(() -> {
               try {
                   authentication();
                   readMessages();
               } catch (IOException e) {
                   e.printStackTrace();
               } finally {
                   closeConnection();
               }
           }).start();
       } catch (IOException e) {
           throw new RuntimeException("Проблемы при создании обработчика
клиента");
       }
   }

   public void authentication() throws IOException {
       while (true) {
           String str = in.readUTF();
           if (str.startsWith("/auth")) {
               String[] parts = str.split("\\s");
               String nick =
myServer.getAuthService().getNickByLoginPass(parts[1], parts[2]);
               if (nick != null) {
                   if (!myServer.isNickBusy(nick)) {
                       sendMsg("/authok " + nick);
                       name = nick;
                       myServer.broadcastMsg(name + " зашел в чат");
                       myServer.subscribe(this);
```

```
                        return;
                    } else {
                        sendMsg("Учетная запись уже используется");
                    }
                } else {
                    sendMsg("Неверные логин/пароль");
                }
            }
        }
    }

    public void readMessages() throws IOException {
        while (true) {
            String strFromClient = in.readUTF();
            System.out.println("от " + name + ": " + strFromClient);
            if (strFromClient.equals("/end")) {
                return;
            }
            myServer.broadcastMsg(name + ": " + strFromClient);
        }
    }

    public void sendMsg(String msg) {
        try {
            out.writeUTF(msg);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void closeConnection() {
        myServer.unsubscribe(this);
        myServer.broadcastMsg(name + " вышел из чата");
        try {
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Написание клиентской части

Много кода связано с графическим интерфейсом — рассмотрим только моменты, связанные с основной логикой чата.
При запуске клиента подключаемся к серверу и попадаем в цикл авторизации, читаем все сообщения с сервера и ожидаем сообщения вида «/authok nick». Как только его получили, переключаем режим авторизации клиента в true, выходим из цикла авторизации и попадаем в цикл общения с сервером. Если пользователь напишет команду «/end», то это сообщение отсылается на серверную сторону, на которой происходит отключение текущего клиента, а на этой стороне(клиента) выходим из цикла общения с сервером и закрываем сокет.

```
try {
    socket = new Socket("localhost", 8189);
    in = new DataInputStream(socket.getInputStream());
    out = new DataOutputStream(socket.getOutputStream());
    setAuthorized(false);
    Thread t = new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                while (true) {
                    String strFromServer = in.readUTF();
                    if(strFromServer.startsWith("/authok")) {
                        setAuthorized(true);
                        break;
                    }
                    chatArea.appendText(strFromServer + "\n");
                }
                while (true) {
                     String strFromServer = in.readUTF();
                     if (strFromServer.equalsIgnoreCase("/end")) {
                         break;
                     }
                     chatArea.append(strFromServer);
                     chatArea.append("\n");
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
    t.setDaemon(true);
    t.start();
} catch (IOException e) {
    e.printStackTrace();
}
```

Метод onAuthClick() отсылает на сервер логин/пароль, введённые в соответствующие поля на клиенте.

```
public void onAuthClick() {
```

```
    try {
        out.writeUTF("/auth " + loginField.getText() + " " + passField.getText());
        loginField.clear();
        passField.clear();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Практическое задание

1. Разобраться с кодом.
2. *Реализовать личные сообщения, если клиент пишет «/w nick3 Привет», то только клиенту с ником nick3 должно прийти сообщение «Привет».

Дополнительные материалы

1 Кей С. Хорстманн, Гари Корнелл. Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. — М.: Вильямс, 2014. — 864 с.
2 Брюс Эккель Философия. Java // 4-е изд.: Пер. с англ. — СПб.: Питер, 2016. — 1 168 с.
3 Г. Шилдт. Java 8. Полное руководство // 9-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 1 376 с.
4 Г. Шилдт. Java 8: Руководство для начинающих. // 6-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 720 с.