|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
|  | |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 3** | |
| **по дисциплине** | |
| **«Архитектура клиент-серверных приложений»** | |
| Выполнил студент группы **ИКБО-10-19** | ***Сороков А.С.*** |
| Принял ассистент кафедры КИС | ***Степанов П.В.*** |

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | « » 2021г. |  |
| «Зачтено» | « » 2021г. |  |

Москва 2021

**Задание**

Необходимо создать клиент-серверное приложение на языке JAVA с использованием socket, для широковещательного общения пользователей. Приложение может быть, как консольным, так и оснащённым полноценным GUI. Клиентское приложение считывает данные из стандартного ввода и отсылает сообщение серверу (с помощью TCP/IP). Сервер, в свою очередь, накапливает сообщения и раз в 5 секунд осуществляет массовую рассылку всем клиентам. Если сообщений за указанный период не поступило, то рассылка не производится. Клиент, получивший сообщение, отображает на экране текст данного сообщения. Структуру и поведение данного клиент-серверного приложения, в том числе, например, в части регистрации конкретного клиента и формата широковещательного сообщения, студент определяет самостоятельно.

**Ход работы**

Разработанная программа состоит из двух частей (серверная и клиентская), которые обмениваются данными посредством socket’ов и протоколу TCP/IP.

Основной класс ClientHandler реализующий сервер показан на листинге:

public class Server {  
 private List<ClientHandler> clients;  
 private static AuthService *authService*;  
 protected static Connection *connection*;  
  
 public Server() {  
 clients = new CopyOnWriteArrayList<>();  
 ServerSocket server = null;  
 Socket socket = null;  
 final int PORT = 8185;  
  
 try {  
 server = new ServerSocket(PORT);  
 System.*out*.println("Server started");  
 *connectDB*();  
 *authService* = new AuthServiceDB();  
  
  
 while (true) {  
 socket = server.accept();  
 new ClientHandler(this, socket);  
 }  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 } finally {  
 try {  
 socket.close();  
 *disconnectDB*();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 try {  
 server.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public void rename(String oldNickname, String newNickname){  
 try {  
 PreparedStatement preparedStatement = *connection*.prepareStatement("UPDATE Users SET nick=? WHERE nick=?");  
 preparedStatement.setString(1,newNickname);  
 preparedStatement.setString(2,oldNickname);  
 preparedStatement.executeUpdate();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
  
 private static void disconnectDB() {  
 try {  
 *connection*.close();  
 } catch (SQLException throwables) {  
 throwables.printStackTrace();  
 }  
 }  
  
 private static void connectDB() throws ClassNotFoundException, SQLException {  
 Class.*forName*("org.sqlite.JDBC");  
 *connection* = DriverManager.*getConnection*("jdbc:sqlite:UsersBD.db");  
 }  
  
 public void broadcastMsg(ClientHandler sender, String msg) {  
 String message = String.*format*("[ %s ]: %s", sender.getNickname(), msg);  
 for (ClientHandler c : clients) {  
 c.sendMsg(message);  
 }  
 }  
  
 public void privateMsg(ClientHandler sender, String receiver, String msg) {  
 String message = String.*format*("[ %s ] private [ %s ] : %s", sender.getNickname(), receiver, msg);  
 for (ClientHandler c : clients) {  
 if (c.getNickname().equals(receiver)) {  
 c.sendMsg(message);  
 if (!sender.getNickname().equals(receiver)) {  
 sender.sendMsg(message);  
 }  
 return;  
 }  
 }  
 sender.sendMsg(String.*format*("Server: Client %s not found", receiver));  
 }  
  
 public void subscribe(ClientHandler clientHandler) {  
 clients.add(clientHandler);  
 broadcastClientList();  
 }  
  
 public void unsubscribe(ClientHandler clientHandler) {  
 clients.remove(clientHandler);  
 broadcastClientList();  
 }  
  
 public AuthService getAuthService() {  
 return *authService*;  
 }  
  
 public boolean isLoginAuthenticated(String login) {  
 for (ClientHandler c : clients) {  
 if (c.getLogin().equals(login)) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 public void broadcastClientList() {  
 StringBuilder sb = new StringBuilder("/clientlist ");  
  
 for (ClientHandler c : clients) {  
 sb.append(c.getNickname()).append(" ");  
 }  
 String message = sb.toString();  
 for (ClientHandler c : clients) {  
 c.sendMsg(message);  
 }  
 }  
}

Также в серверной части приложения реализован класс ClientHandler для общения с клиентской частью через сокетов. Данный класс продемонстрирован на листинге:

public class ClientHandler {  
 DataInputStream in;  
 DataOutputStream out;  
 Server server;  
 Socket socket;  
  
 private String nickname;  
 private String login;  
  
 public ClientHandler(Server server, Socket socket) {  
 try {  
 this.server = server;  
 this.socket = socket;  
 in = new DataInputStream(socket.getInputStream());  
 out = new DataOutputStream(socket.getOutputStream());  
 System.*out*.println("Client connected " + socket.getRemoteSocketAddress());  
  
 new Thread(() -> {  
 try {  
 socket.setSoTimeout(50000);  
  
 //цикл аутентификации  
 while (true) {  
 String str = in.readUTF();  
  
 if (str.startsWith("/reg ")) {  
 String[] token = str.split("\\s");  
 if (token.length < 4) {  
 continue;  
 }  
 boolean b = server.getAuthService()  
 .registration(token[1], token[2], token[3]);  
 if (b) {  
 sendMsg("/regok");  
 } else {  
 sendMsg("/regno");  
 }  
 }  
  
 if (str.startsWith("/auth ")) {  
 String[] token = str.split("\\s");  
 if (token.length < 3) {  
 continue;  
 }  
 String newNick = server.getAuthService()  
 .getNicknameByLoginAndPassword(token[1], token[2]);  
 if (newNick != null) {  
 login = token[1];  
 if (!server.isLoginAuthenticated(login)) {  
 nickname = newNick;  
 sendMsg("/authok " + newNick);  
 getLastMessage();  
 server.subscribe(this);  
 socket.setSoTimeout(0);  
 break;  
 } else {  
 sendMsg("С этим логином уже вошли в чат");  
 }  
 } else {  
 sendMsg("Неверный логин / пароль");  
 }  
 }  
 }  
  
 //цикл работы  
 while (true) {  
 String str = in.readUTF();  
 if (str.startsWith("/")) {  
 if (str.equals("/end")) {  
 sendMsg("/end");  
 break;  
 }  
 if (str.startsWith("/w ")) {  
 String[] token = str.split("\\s", 3);  
 if (token.length < 3) {  
 continue;  
 }  
 server.privateMsg(this, token[1], token[2]);  
 }  
 String[] subStr = str.split(" ");  
 if (subStr[0].equals("/renickname")){  
 server.rename(nickname, subStr[1]);  
 nickname = subStr[1];  
 }  
 } else {  
 server.broadcastMsg(this, str);  
 }  
 }  
  
 } catch (SocketTimeoutException e) {  
 sendMsg("/end");  
 System.*out*.println("Client disconnected by timeout");  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 server.unsubscribe(this);  
 System.*out*.println("Client disconnected " + socket.getRemoteSocketAddress());  
 try {  
 socket.close();  
 in.close();  
 out.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void getLastMessage() throws IOException {  
 List<String> tmp = new ArrayList<String>();  
 String strpath="LogServer.txt";  
 FileReader fr = null;  
 try {  
 fr = new FileReader(strpath);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 BufferedReader br = new BufferedReader(fr);  
 String ch;  
  
 do {  
 ch = br.readLine();  
 tmp.add(ch);  
 } while (ch != null);  
  
 int limiter = 0;  
 for(int i=tmp.size()-2;i>=0;i--) {  
 limiter++;  
 if(limiter <= 20) {  
 sendMsgNoLog(tmp.get(i));  
 System.*out*.println(tmp.get(i));  
 }  
 else{  
 break;  
 }  
 }  
  
 }  
  
  
 public void sendMsg(String msg) {  
 try {  
 out.writeUTF(msg);  
 writeLog(msg);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void sendMsgNoLog(String msg) {  
 try {  
 out.writeUTF(msg);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void writeLog(String msg){  
 if(!msg.startsWith("/")){  
 try(FileWriter writer = new FileWriter("LogServer.txt", true))  
 {  
 writer.write(msg + "\n");  
 writer.flush();  
 }  
 catch(IOException ex){  
 System.*out*.println(ex.getMessage());  
 }  
 }  
 }  
  
 public String getNickname() {  
 return nickname;  
 }  
  
 public String getLogin() {  
 return login;  
 }  
}

Клиентскую часть приложения можно рассмотреть на листинге:

public class Controller implements Initializable {  
 @FXML  
 private ListView<String> clientList;  
 @FXML  
 private TextArea textArea;  
 @FXML  
 private TextField textField;  
 @FXML  
 private HBox authPanel;  
 @FXML  
 private TextField loginField;  
 @FXML  
 private PasswordField passwordField;  
 @FXML  
 private HBox msgPanel;  
  
 private final String IP\_ADDRESS = "localhost";  
 private final int PORT = 8185;  
  
 private Socket socket;  
 private DataInputStream in;  
 private DataOutputStream out;  
  
 private Stage stage;  
 private Stage regStage;  
 private RegController regController;  
  
 private boolean authenticated;  
 private String nickname;  
  
 private void setAuthenticated(boolean authenticated) {  
 this.authenticated = authenticated;  
 authPanel.setVisible(!authenticated);  
 authPanel.setManaged(!authenticated);  
 msgPanel.setVisible(authenticated);  
 msgPanel.setManaged(authenticated);  
 clientList.setVisible(authenticated);  
 clientList.setManaged(authenticated);  
  
 if (!authenticated) {  
 nickname = "";  
 setTitle("Балабол");  
 } else {  
 setTitle(String.*format*("[ %s ] - Балабол", nickname));  
 }  
 textArea.clear();  
  
 }  
  
 @Override  
 public void initialize(URL location, ResourceBundle resources) {  
 Platform.*runLater*(() -> {  
 stage = (Stage) textField.getScene().getWindow();  
 stage.setOnCloseRequest(event -> {  
 System.*out*.println("bye");  
 if (socket != null && !socket.isClosed()) {  
 try {  
 out.writeUTF("/end");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
 });  
 setAuthenticated(false);  
 createRegWindow();  
 }  
  
  
 private void connect() {  
 try {  
 socket = new Socket(IP\_ADDRESS, PORT);  
 in = new DataInputStream(socket.getInputStream());  
 out = new DataOutputStream(socket.getOutputStream());  
  
 new Thread(() -> {  
 try {  
 //цикл аутентификации  
 while (true) {  
 String str = in.readUTF();  
  
 if (str.startsWith("/authok ")) {  
 nickname = str.split("\\s")[1];  
 setAuthenticated(true);  
 break;  
 }  
  
 if (str.startsWith("/regok")) {  
 regController.addMessageTextArea("Регистрация прошла успешно");  
 }  
 if (str.startsWith("/regno")) {  
 regController.addMessageTextArea("Зарегистрироватся не удалось\n" +  
 " возможно такой логин или никнейм уже заняты");  
 }  
  
 if (str.startsWith("/end")) {  
 throw new RuntimeException("disconnected by timeout");  
 }  
  
 textArea.appendText(str + "\n");  
 }  
  
 //цикл работы  
 while (true) {  
 String str = in.readUTF();  
  
 if (str.startsWith("/")) {  
 if (str.equals("/end")) {  
 break;  
 }  
 if (str.startsWith("/clientlist ")) {  
 String[] token = str.split("\\s");  
 Platform.*runLater*(() -> {  
 clientList.getItems().clear();  
 for (int i = 1; i < token.length; i++) {  
 clientList.getItems().add(token[i]);  
 }  
 });  
 }  
 } else {  
 textArea.appendText(str + "\n");  
 writeToFile(nickname,str);  
  
 }  
 }  
 } catch (RuntimeException e) {  
 System.*out*.println(e.getMessage());  
 } catch (IOException e) {  
 e.printStackTrace();  
 } finally {  
 setAuthenticated(false);  
 try {  
 socket.close();  
 in.close();  
 out.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void writeToFile(String nickname,String message){  
 try(FileWriter writer = new FileWriter(nickname+".txt", true))  
 {  
 writer.write(message + "\n");  
 writer.flush();  
 }  
 catch(IOException ex){  
  
 System.*out*.println(ex.getMessage());  
 }  
  
 }  
  
 public void sendMsg(ActionEvent actionEvent) {  
 if (textField.getText().trim().length() == 0) {  
 return;  
 }  
 try {  
 out.writeUTF(textField.getText());  
 textField.clear();  
 textField.requestFocus();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void tryToAuth(ActionEvent actionEvent) {  
 if (socket == null || socket.isClosed()) {  
 connect();  
 }  
  
 String msg = String.*format*("/auth %s %s",  
 loginField.getText().trim(), passwordField.getText().trim());  
 try {  
 out.writeUTF(msg);  
 passwordField.clear();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private void setTitle(String title) {  
 Platform.*runLater*(() -> {  
 stage.setTitle(title);  
 });  
 }  
  
 public void clickClientList(MouseEvent mouseEvent) {  
 textField.setText(String.*format*("/w %s ", clientList.getSelectionModel().getSelectedItem()));  
 }  
  
 public void releasedMouseClientList(MouseEvent mouseEvent) {  
 System.*out*.println(clientList.getSelectionModel().getSelectedItem());  
 System.*out*.println(mouseEvent.getButton());  
 System.*out*.println(mouseEvent.getClickCount());  
 }  
  
 private void createRegWindow() {  
 try {  
 FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("/reg.fxml"));  
 Parent root = fxmlLoader.load();  
 regStage = new Stage();  
 regStage.setTitle("Регистрация в чате Балабол");  
 regStage.setScene(new Scene(root, 400, 300));  
 regStage.initModality(Modality.*APPLICATION\_MODAL*);  
  
 regController = fxmlLoader.getController();  
 regController.setController(this);  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void regStageShow(ActionEvent actionEvent) {  
 regStage.show();  
 }  
  
 public void tryRegistration(String login, String password, String nickname) {  
 String msg = String.*format*("/reg %s %s %s", login, password, nickname);  
  
 if (socket == null || socket.isClosed()) {  
 connect();  
 }  
  
 try {  
 out.writeUTF(msg);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
  
}

Выполненный проект можно посмотреть по ссылке: https://github.com/sorokow/java\_3\_chat\_23102020

**Вывод**

В ходе практической работы мы познакомились с протоколом TCP/IP и научились создавать клиент-серверные приложения на основе сокетов.