|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №3**

по дисциплине «Архитектура клиент-серверных приложений»

**Тема практической работы:** Программирование JAVA сокетов

**Студент группы** ИКБО-01-19 Кузин Д.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** ассистент Китанин С.С.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Москва 2021

# Цель

Знакомство студентов с одной из базовых технологии реализации конечной точки для передачи и приема данных по сети - сокетом.

# Задание на практическую работу

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

В ходе работы были созданы две программы: программа сервера и программа клиента. В процессе были использованы сокеты и потоки. Код программы-сервера представлен на листинге 1.

Листинг 1 – Файл Server.java

public class Server {  
 private static final StringBuilder *inputMessage* = new StringBuilder();  
 private static final StringBuilder *inputMessageOld* = new StringBuilder();  
 private ServerSocket serverSocket;  
 static final List<PrintWriter> *printWriters* = new ArrayList<>();  
  
  
 public void start(int port) throws IOException {  
 serverSocket = new ServerSocket(port);  
 ExecutorService pool = Executors.*newFixedThreadPool*(20);  
 mailing();  
 while (true) {  
 pool.execute(new MailingHandler(serverSocket.accept()));  
 }  
 }  
  
 private void mailing() {  
 Timer carousalTimer = new Timer();  
 carousalTimer.scheduleAtFixedRate(new TimerTask() {  
 @Override  
 public void run() {  
 if (*inputMessage*.length() > 0) {  
 *inputMessageOld*.setLength(0);  
 *inputMessageOld*.append(*inputMessage*);  
 *inputMessage*.setLength(0);  
 for (PrintWriter out : *printWriters*) {  
 System.*out*.println(*inputMessageOld*);  
 out.println(*inputMessageOld*);  
 }  
 }  
 }  
 }, 5000, 5000);  
 }  
  
 public void stop() throws IOException {  
 serverSocket.close();  
 }  
  
  
 private static class MailingHandler extends Thread {  
 private final Socket clientSocket;  
 private PrintWriter out;  
 private BufferedReader in;  
  
  
 public MailingHandler(Socket socket) {  
 this.clientSocket = socket;  
 }  
  
 public void run() {  
 try {  
 out = new PrintWriter(clientSocket.getOutputStream(), true);  
 in = new BufferedReader(  
 new InputStreamReader(clientSocket.getInputStream()));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 if (out != null) {  
 *printWriters*.add(out);  
 }  
  
 while (true) {  
 String mes = "";  
 try {  
 mes = in.readLine();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 if (".".equals(mes)) {  
 out.println("bye");  
 try {  
 Thread.*sleep*(500);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 break;  
 }  
 if (!"".equals(mes)) {  
 *inputMessage*.append(mes).append(" ");  
 }  
 }  
 *printWriters*.remove(out);  
 out.close();  
 try {  
 in.close();  
 clientSocket.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
 }  
  
 public static void main(String[] args) throws IOException {  
 new Server().start(8221);  
 }  
}

Код программы-клиента, отправляющей сообщения серверу и получающей от него все отправленные сообщение каждые 5 секунд представлен на листинге 2.

Листинг 2 – Файл Client.java

public class Client {  
 private Socket clientSocket;  
 private PrintWriter out;  
 private BufferedReader in;  
  
 public void startConnection(String ip, int port) throws IOException {  
 clientSocket = new Socket(ip, port);  
 out = new PrintWriter(clientSocket.getOutputStream(), true);  
 in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
 }  
  
 public void sendMessage(String msg) {  
 out.println(msg);  
 }  
  
 public void getMessage() {  
 new Thread(() -> {  
 while (true) {  
 try {  
 String response = in.readLine();  
 System.*out*.println(response);  
 if("bye".equals(response)){  
 stopConnection();  
 break;  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }).start();  
 }  
  
 public void stopConnection() throws IOException {  
 in.close();  
 out.close();  
 clientSocket.close();  
 }  
  
 public static void main(String[] args) throws IOException, InterruptedException {  
 Client client = new Client();  
 client.startConnection("localhost", 8221);  
 client.getMessage();  
  
 Scanner sn = new Scanner(System.*in*);  
 while (true) {  
 String message = sn.nextLine();  
 client.sendMessage(message);  
 if(".".equals(message)){  
 break;  
 }  
 }  
  
 }  
  
}

Результаты работы программы представлены на рисунках 1-2.

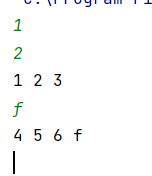


Рисунок – Результат работы программы со стороны первого клиента

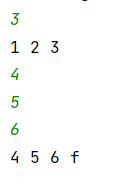


Рисунок – Результат работы программы со стороны второго клиента

# Вывод

В результате работы была изучена одна из базовых технологий реализации конечной точки для передачи и приема данных по сети – сокетом на примере клиент-серверного приложения