

Лабораторная работа №6

Технология программирования

*«Сетевые “клиент-серверные” приложения»*

Студент: Молнер В.С.

Группа: АВТ-814

Вариант: 4

Преподаватель: Михайленко Д.А.



НОВОСИБИРСК 2020

Оглавление

[Задание 1](#_Toc38223051)

[Ход работы 1](#_Toc38223052)

[Вывод 1](#_Toc38223053)

# Задание

1. Изучить особенности реализации сетевых приложений в Java.
2. Доработать программу, созданную в лабораторной работе № 5:

* создать отдельное консольное приложение, которое будет играть роль TCP-сервера. Клиентом будет приложение, которое было создано в предыдущих работах;
* TCP-сервер должен ожидать подключения клиентов и выдавать вновь подключенному клиенту список уже подключенных. Необходимо также внести изменения в интерфейс клиента, так чтобы в панели управления отображался список всех подключенных к серверу клиентов. При отсоединении клиентов или при подключении новых список должен обновляться;
* запрограммировать специальное взаимодействие по TCP с другими клиентами через сервер по варианту;

**Вариант 4**

Реализовать возможность передачи своих настроек симуляции одному из подключенных клиентов.

# Ход работы

## NetWork

Для общения между клиентом и сервером был создан модуль NetWork – абстракция на связь. В него были добавлены класс TCPConnection и интерфейс TCPConnectionListener. Класс описывает канал связи и хранит в себе сокет, потоки ввода-вывода и объект интерфейса TCPConnectionListener. Главные метод этого класса – отправить сообщение (sendString).

public class TCPConnection {  
  
 private final Socket socket;  
 private Thread rxThread = null;  
 private final TCPConnectionListener eventListener;  
 private final BufferedReader in;  
 private final BufferedWriter out;  
  
 public TCPConnection(TCPConnectionListener eventListener, String ip, int port) throws IOException {  
 this(eventListener, new Socket(ip, port));  
 }  
  
 public TCPConnection(TCPConnectionListener eventListener, Socket socket) throws IOException {  
 this.eventListener = eventListener;  
 this.socket = socket;  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 out = new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()));  
 rxThread = new Thread(() -> {  
 try {  
 eventListener.onConnectionReady(TCPConnection.this);  
 while (!rxThread.isInterrupted()) {  
 eventListener.onReceiveString(TCPConnection.this, in.readLine());  
 }  
 } catch (IOException e) {  
 eventListener.onException(TCPConnection.this, e);  
 } finally {  
 eventListener.onDisconnection(TCPConnection.this);  
 }  
 });  
 rxThread.start();  
 }  
  
 public synchronized void sendString(String value) {  
 try {  
 out.write(value + "\r\n");  
 out.flush();  
 } catch (IOException e) {  
 eventListener.onException(TCPConnection.this, e);  
 disconnect();  
 }  
 }  
  
 public synchronized void disconnect() {  
 rxThread.interrupt();  
 try {  
 socket.close();  
 } catch (IOException e) {  
 eventListener.onException(TCPConnection.this, e);  
 }  
 }  
  
 @Override  
 public String toString() {  
 return socket.getPort() + "";  
 }  
  
 public int getSocketPort() { return socket.getLocalPort(); }  
  
 public int getServerSocketPort() { return socket.getPort(); }  
  
}

В свою очередь интерфейс TCPConnectionListner является абстракцией слушателя событий. Он хранит в себе сигнатуры четырёх методов: подключился, отключился, получил сообщение и поймал ошибку.

public interface TCPConnectionListener {  
  
 void onConnectionReady(TCPConnection tcpConnection);  
 void onReceiveString(TCPConnection tcpConnection, String value);  
 void onDisconnection(TCPConnection tcpConnection);  
 void onException(TCPConnection tcpConnection, Exception e);  
  
}

Благодаря такой абстракции связи появилась возможность реализовать соединение со стороны клиента и со стороны сервера по одному подобию.

## Server

Следующим шагом был создан сервер. Он реализует интерфейс TCPConnectionListner и по сути является слушателем событий.

public class Server implements TCPConnectionListener{  
  
 public static void main(String[] args) {  
 new Server();  
  
 }  
  
 private final HashMap<Integer,TCPConnection> connections = new HashMap<>();  
  
 private Server() {  
 System.*out*.println("Server is running...");  
 try (ServerSocket serverSocket = new ServerSocket(3333)) {  
 while (true) {  
 try {  
 new TCPConnection(this, serverSocket.accept());  
 } catch (IOException e) {  
 System.*out*.println("TCPConnection exception: " + e);  
 }  
 }  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
  
 @Override  
 public synchronized void onConnectionReady(TCPConnection tcpConnection) {  
 connections.put(tcpConnection.getServerSocketPort(),tcpConnection);  
 sendToAllConnections();  
 }  
  
 @Override  
 public synchronized void onReceiveString(TCPConnection tcpConnection, String value) {  
 if (value.substring(0, 8).equals("Request ")) {  
 int key = Integer.*parseInt*(value.substring(8));  
 if (connections.get(key) != null) {  
 connections.get(key).sendString("Config " + tcpConnection.getServerSocketPort());  
 }  
 } else if (value.substring(0, 8).equals("Command ")) {  
 int key = Integer.*parseInt*(value.substring(8,value.indexOf(' ',8)));  
 if (connections.get(key) != null) {  
 connections.get(key).sendString("File " + value.substring(value.indexOf(' ',9)));  
 }  
 }  
 }  
  
 @Override  
 public synchronized void onDisconnection(TCPConnection tcpConnection) {  
 connections.remove(tcpConnection.getServerSocketPort());  
 sendToAllConnections();  
 }  
  
 @Override  
 public synchronized void onException(TCPConnection tcpConnection, Exception e) {  
 System.*out*.println("TCPConnection exception: " + e);  
 }  
  
 private void sendToAllConnections() {  
 connections.forEach((key,tcpConnection) ->  
 connections.forEach((key1,tcpConnection1) ->  
 tcpConnection.sendString("User " + tcpConnection1.toString())));  
 }  
}

В конструкторе сервера находится бесконечный цикл.

## Client

public class Client implements TCPConnectionListener {  
  
 private static final String *IP* = "localhost";  
 private static final int *port* = 3333;  
 private static TCPConnection *tcpConnection* = null;  
 private static Vector<String> *UsersOnline* = new Vector<>();  
  
 public Client() {  
 System.*out*.println("Client is running...");  
 try {  
 *tcpConnection* = new TCPConnection(this, *IP*, *port*);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void RequestConfigs(String SocketPort) {  
 *tcpConnection*.sendString("Request " + SocketPort);  
 }  
  
 @Override  
 public void onConnectionReady(TCPConnection tcpConnection) {  
 System.*out*.println("Connection ready");  
 }  
  
 @Override  
 public void onReceiveString(TCPConnection tcpConnection, String value) {  
 System.*out*.println(value);  
 if (value.substring(0,7).equals("Config ")) {  
 File file = new File("Configuration.txt");  
 try (Scanner scanner = new Scanner(file)) {  
 String string = scanner.nextLine();  
 tcpConnection.sendString("Command " + value.substring(7) + " " + string);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 } else if (value.substring(0,5).equals("File ")) {  
 File file = new File("Configuration.txt");  
 try (FileWriter fileWriter = new FileWriter(file)) {  
 fileWriter.write(value.substring(6));  
 fileWriter.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 } else if (value.substring(0,5).equals("User ")) {  
 if (!(*UsersOnline*.indexOf(value.substring(5)) >= 0) && Integer.*parseInt*(value.substring(5)) != tcpConnection.getSocketPort())  
 *UsersOnline*.add(value.substring(5));  
 }  
 }  
  
 @Override  
 public void onDisconnection(TCPConnection tcpConnection) {  
 System.*out*.println("Disconnected");  
 }  
  
 @Override  
 public void onException(TCPConnection tcpConnection, Exception e) {  
 System.*out*.println("Connection exception: " + e);  
 }  
  
 public Vector<String> getUsersOnline() {  
 return *UsersOnline*;  
 }  
  
}

## Общее

Было создано отдельное немодальное диалоговое окно для клиента. Оно открывается из главного меню. В окне находится JList, в котором показываются пользователи онлайн. Если пользователи не показываются нужно нажать клавишу U и список обновится. Чтобы получить настройки одного из пользователей, нужно щёлкнуть на него мышкой и нажать Enter.

# Вывод

В процессе выполнения лабораторной работы произошло знакомство с протоколом TCP. Также были более тщательно изучены потоки ввода-вывода. Была доработана программа из лабораторной работы №5.