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

Институт информационных технологий

КАФЕДРА ИНСТРУМЕНТАЛЬНОГО И ПРИКЛАДНОГО
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИиППО)

Практическая работа №3
«Программирование Java сокетов»

По дисциплине: «Архитектура клиент-серверных приложений»

Выполнил студент группы ИКБО-10-19 Дараган Ф.А.

Принял преподаватель Степанов П.В.

Практическая работы выполнена «___»_____2021 г.
(подпись студента)

«Зачтено» «___»_____2021 г.
(подпись руководителя)

Москва 2021

Оглавление

Практическая работа № 3 Программирование Java сокетов

Цель работы

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

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.

Выполнение практической работы

На рисунке 1 представлена иерархия файлов проекта.

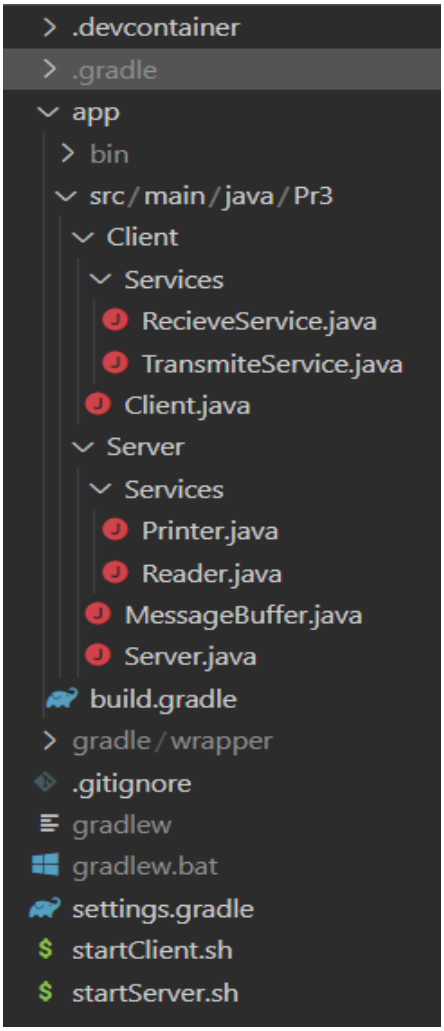

Рис. 1. Скриншот иерархии файлов проекта

На листинге 1 представлен RecieveService, реализующий передачу сообщений от клиента к серверу.

Листинг 1. Класс RecieveService

```
package Pr3.Client.Services;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import Pr3.Client.Client;
```

```
public class RecieveService implements Runnable {

    private BufferedReader in;
    private Client client;

    public RecieveService(Client client) throws IOException{
        this.client = client;
        this.in = new BufferedReader(new
InputStreamReader(client.getSocket().getInputStream()));
    }

    @Override
    public void run() {
        while (!Thread.interrupted()) {
            try {
                var str = in.readLine();
                if (str == null) {
                    System.out.println("RecieveService stopped, where
client = " + client.getName());
                    break;
                }
                System.out.println(str);
            } catch (IOException ignore) {}
        }
    }

}
```

На листинге 2 находится TransmiteService, осуществляющий прием сообщений от сервера и выводящий их на экран.

Листинг 2. Класс TransmiteService

```
package Pr3.Client.Services;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;

import Pr3.Client.Client;

public class TransmiteService implements Runnable {

    private BufferedWriter out;
    private BufferedReader in;
    private Client client;

    public TransmiteService(Client client) throws IOException{
```

```
        this.client = client;
        this.out = new BufferedWriter(new
OutputStreamWriter(client.getSocket().getOutputStream()));
        this.in = client.getInput();
    }

    @Override
    public void run() {
        while (!Thread.interrupted()) {
            try {
                var str = in.readLine();
                out.write(client.getName() + ": " + str + "\n");
                out.flush();
            } catch (IOException e) {
                System.out.println("TransmiteService stopped, where
client = " + client.getName());
                break;
            }
        }
    }

}
```

На листинге 3 представлен Client, класс реализующий клиентскую логику и объединяющий RecieveService и TransmiteService. Метод makeClient был опущен, ибо не важен для реализации работы с сокетами и потоками.

Листинг 3. Класс Client

```
package Pr3.Client;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.IOException;
import java.net.Socket;

import Pr3.Client.Services.RecieveService;
import Pr3.Client.Services.TransmiteService;

public class Client {

    private Socket socket;
    private String name;
    private BufferedReader in;
    private BufferedWriter out;
    private Thread reciever;
    private Thread transmitter;
```

```
    public static Client makeClient(BufferedReader in, BufferedWriter
out) throws IOException, Error;

    public Client(String name, String addr, int port, BufferedReader
in, BufferedWriter out) throws Error {
        this.name = name;
        this.in = in;
        this.out = out;
        try {
            this.socket = new Socket(addr, port);
            startServices();
            System.out.println("Client started...");
        } catch (IOException e) {
            throw new Error("Client initialization failed", e);
        }
    }

    public Socket getSocket() {
        return this.socket;
    }

    public BufferedReader getInput() {
        return this.in;
    }

    public BufferedWriter getOutput() {
        return this.out;
    }

    public String getName() {
        return this.name;
    }

    public void startServices() throws IOException {
        this.reciever = new Thread(new RecieveService(this));
        this.transmitter = new Thread(new TransmiteService(this));
        reciever.start();
        transmitter.start();
    }

    public static void main(String[] args) {
        var in = new BufferedReader(new
InputStreamReader(System.in));
        var out = new BufferedWriter(new
OutputStreamWriter(System.out));
        try {
            Client.makeClient(in, out);
        } catch (IOException e) {
            e.printStackTrace();
```

```
        } catch (Error e) {
            e.printStackTrace();
        }
    }

}
```

На листинге 4 представлен вспомогательный класс для хранения сообщений на сервере.

Листинг 4. Класс MessageBuffer

```
package Pr3.Server;

import java.util.LinkedList;
import java.util.List;

public class MessageBuffer {
    public static List<String> messages = new LinkedList<>();

    public static void addMessage(String m) {
        synchronized (messages) {
            messages.add(m);
        }
    }

    public static List<String> injectMessages() {
        List<String> ret;
        synchronized (messages) {
            ret = messages;
            messages = new LinkedList<>();
        }
        return ret;
    }
}
```

На листинге 5 представлен класс Printer, реализующий рассылку сообщений с сервера клиентам раз в 5 секунд, если за это время были новые сообщения.

Листинг 5. Класс Printer

```
package Pr3.Server.Services;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.LinkedList;
import java.util.List;
import java.util.stream.Collectors;

import Pr3.Server.MessageBuffer;
```

```
public class Printer implements Runnable {

    private List<BufferedWriter> consumers;

    public Printer() {
        consumers = new LinkedList<>();
    }

    public void addConsumer(BufferedWriter consumer) {
        synchronized (consumers) {
            consumers.add(consumer);
        }
    }

    public void removeConsumer(BufferedWriter consumer) {
        synchronized (consumers) {
            consumers.remove(consumer);
        }
    }

    @Override
    public void run() {
        while (!Thread.interrupted()) {
            try {
                Thread.sleep(5000);
            } catch (InterruptedException ignore) {
                System.out.println("Printer::run Sleep problem");
            }

            List<String> messages = MessageBuffer.injectMessages();

            if (messages.size() == 0) {
                System.out.println("No messages");
                continue;
            }

            var res = "--------------New Messages--------------\n" +
                      messages.stream().collect(Collectors.joining("\
n")) + "\n" +
                      "----------------------------------------\n";

            synchronized (consumers) {
                var consumersCopy = consumers;
                consumersCopy.stream().forEach(
                    (consumer) -> {
                        try {
                            consumer.write(res);
                            consumer.flush();
                        } catch (IOException ignore) {
```

```
                            System.out.println("Consumer removed");
                            consumers.remove(consumer);
                        }
                    }
                );
            }
        }
    }

}
```

На листинге 6 представлен класс Reader, получающий сообщения от пользователей и сохраняющий их в MessageBuffer.

Листинг 6. Класс Reader

```
package Pr3.Server.Services;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;

import Pr3.Server.MessageBuffer;

public class Reader implements Runnable {

    private BufferedReader in;
    private BufferedWriter out;

    public Reader(BufferedReader in, BufferedWriter out) {
        this.in = in;
        this.out = out;
    }

    @Override
    public void run() {
        while (!Thread.interrupted()) {
            try {
                var text = in.readLine();
                if (text == null) {
                    System.out.println("Reader stopped");
                    break;
                }
                MessageBuffer.addMessage(text);
                out.write(text + "\n");
                out.flush();
            } catch (IOException e) {

            }
        }
```

```
    }

}
```

На листинге 7 показан класс Сервера, осуществляющий аккумуляцию сообщений и их рассылку, с помощью классов Reader и Printer.

Листинг 7. Класс Reader

```
package Pr3.Server;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.BufferedWriter;
import java.io.IOException;
import java.net.ServerSocket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.net.Socket;

import Pr3.Server.Services.Printer;
import Pr3.Server.Services.Reader;

public class Server {

    private Thread printer;

    Server(int port) {

        ExecutorService clients = Executors.newFixedThreadPool(20);
        ServerSocket server;

        try {
            server = new ServerSocket(port);
        } catch (IOException e1) {
            throw new Error("Can't up server", e1);
        }

        Printer printer = new Printer();
        this.printer = new Thread(printer);
        this.printer.start();

        System.out.println("Server started...");

        try {
            while (true) {
                Socket socket = server.accept();
                var in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
```

```
                var out = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream()));
                var sysOut = new BufferedWriter(new
OutputStreamWriter(System.out));
                clients.execute(new Reader(in, sysOut));
                printer.addConsumer(out);
            }
        } catch (IOException ignore) {}
        finally {
            try {
                server.close();
            } catch (IOException e) {
                throw new Error("Can't down server", e);
            }
            this.printer.interrupt();
            clients.shutdown();
        }
    }

    public static void main(String[] args) {
        new Server(8080);
    }

}
```

Для запуска клиента и сервера написаны 2 скрипта, код представлен на листингах 8 и 9.

Листинг 8. Скрипт startServer.sh

```
#!/bin/bash

/usr/bin/env /usr/local/openjdk-16/bin/java -XX:
+ShowCodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -cp
/workspaces/GitRep/Pr3/Code/app/bin/main Pr3.Server.Server
```

Листинг 9. Скрипт startClient.sh

```
#!/bin/bash

/usr/bin/env /usr/local/openjdk-16/bin/java -XX:
+ShowCodeDetailsInExceptionMessages -Dfile.encoding=UTF-8 -cp
/workspaces/GitRep/Pr3/Code/app/bin/main Pr3.Client.Client
```

Для работы приложения необходимо сначала запустить сервер и лишь потом клиент. Результат выполнения программы представлен на рисунках 2, 3, 4.

```
vscode ➜/workspaces/GitRep/Pr3/Code (master ✗) $ ./startServer.sh
Server started...
No messages
u1: Hello all
No messages
u2: Hi
u1: IM u1
u2: Im u2
Reader stopped
Reader stopped
No messages
^Cvscode ➜/workspaces/GitRep/Pr3/Code (master ✗) $
```

Рис. 2. Скриншот запуска сервера

```
vscode ➜/workspaces/GitRep/Pr3/Code (master ✗) $ ./startClient.sh
Enter your name: u1
Enter address [default localhost]:
Enter port [default 8080]:
Client started...
Hello all
--------------New Messages--------------
u1: Hello all
----------------------------------------
--------------New Messages--------------
u2: Hi
----------------------------------------
IM u1
--------------New Messages--------------
u1: IM u1
----------------------------------------
--------------New Messages--------------
u2: Im u2
----------------------------------------
^Cvscode ➜/workspaces/GitRep/Pr3/Code (master ✗) $
```

Рис. 3. Скриншот запуска клиента u1

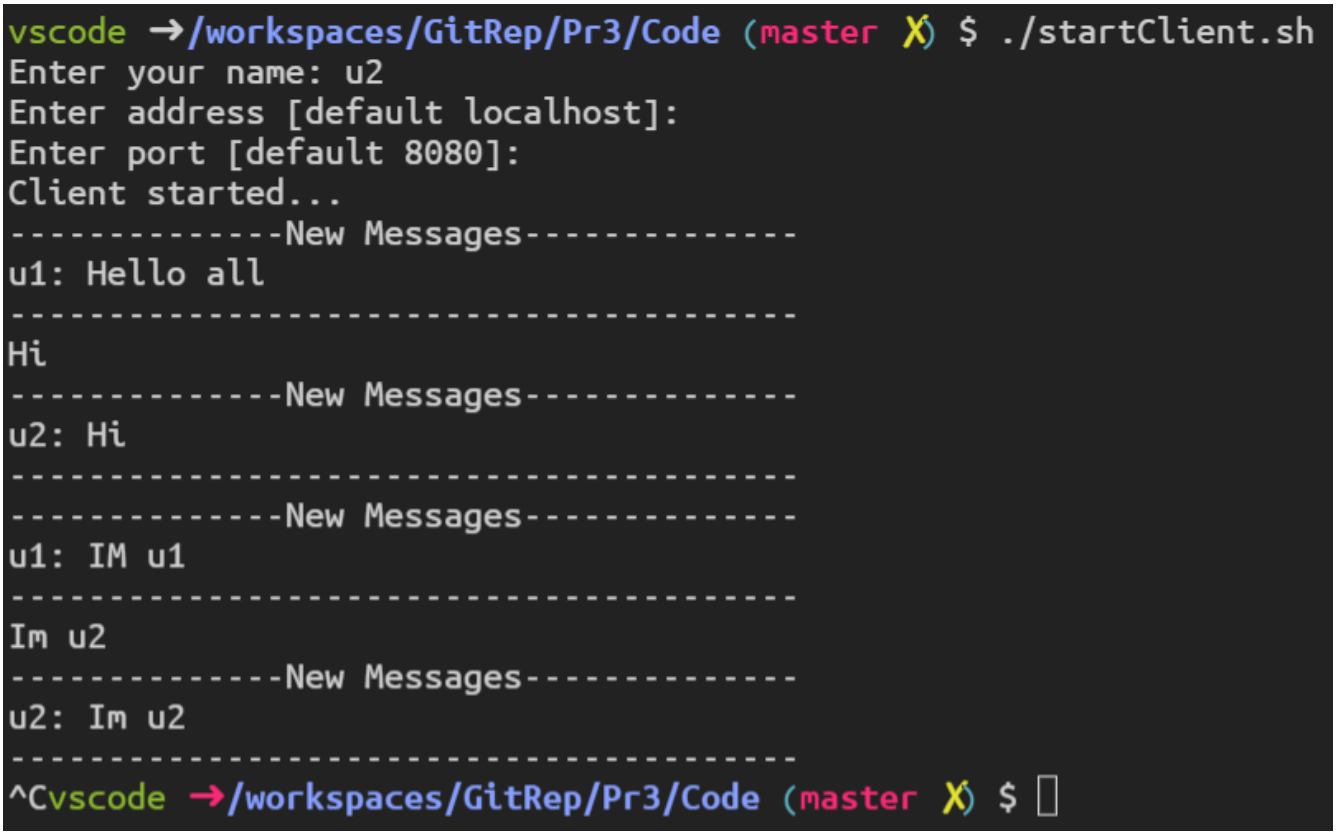

Рис. 4. Скриншот запуска клиента u2

Выводы по работе

Мы научились работать с Java сокетами, построив простейшую серверную конфигурацию с их помощью. Так же в процессе выполнения работы были задействованы потоки и пулы потоков для эффективной реализации работы приложения.

Используемая литература

1. Вязовик, Н. А. Программирование на Java : учебное пособие / Н. А. Вязовик. — 2-е изд. — Москва : ИНТУИТ, 2016. — 603 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/100405 (дата обращения: 13.09.2021). — Режим доступа: для авториз. пользователей.
2. Наир, В. Предметно-ориентированное проектирование в Enterprise Java : руководство / В. Наир ; перевод с английского А. В. Снастина. — Москва : ДМК Пресс, 2020. — 306 с. — ISBN 978-5-97060-872-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/179503 (дата обращения: 13.09.2021). — Режим доступа: для авториз. пользователей.
3. Васильев, А. Н. Самоучитель Java с примерами и программами : учебное пособие / А. Н. Васильев. — 4-е, изд. — Санкт-Петербург : Наука и Техника, 2017. — 368 с. — ISBN 978-5-94387-745-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/101548 (дата обращения: 13.09.2021). — Режим доступа: для авториз. пользователей.