Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №6

по курсу «Программирование на языке Java»

на тему «Сетевое взаимодействие в Java»

Выполнили:

студенты группы 20ВВП1

Войнова Д.А.

Зиновьев Я.М.

Проверили:

Юрова О.В.

Карамышева Н.С.

Пенза 2023

**Цель работы:** научиться создавать клиент-серверные приложения c использованием стандартных классов Java.

**Задание:**

Модифицировать приложение из предыдущей лабораторной работы, реализовав клиент-серверную архитектуру, обеспечивающую распределенное вычисление определенного интеграла на нескольких вычислительных узлах (клиентах) при этом каждый узел использует несколько нитей, как в предыдущей работе. Сервер не занимается вычислениями, а лишь реализует взаимодействие с пользователем и агрегацию результатов вычислений от клиентов.

3 вариант - протокол UDP

Результат работы программы:

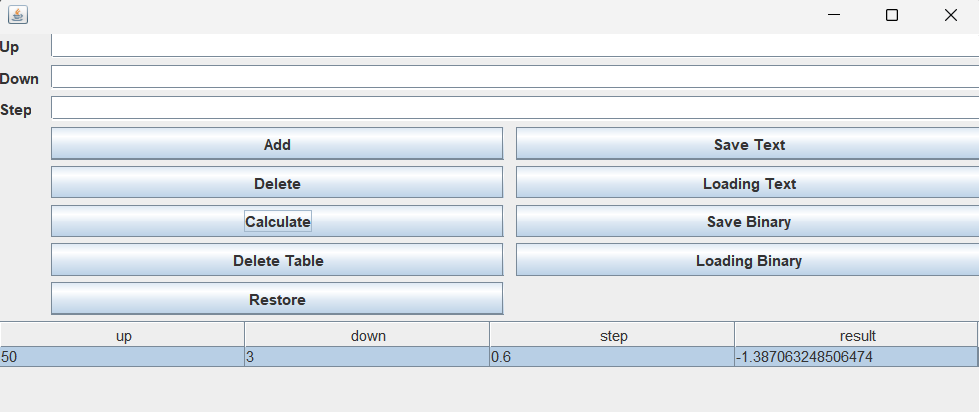


Рис.1

Листинг

Сервер

app.java

package main.java;

import javax.swing.\*;

import javax.swing.event.TableModelEvent;

import javax.swing.event.TableModelListener;

import javax.swing.filechooser.FileSystemView;

import javax.swing.table.DefaultTableModel;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.\*;

import java.net.SocketException;

import java.util.\*;

public class App extends JFrame {

private JTextField up;

private JTextField step;

private JTextField down;

private JButton deleteButton;

private JButton addButton;

private JButton calculateButton;

private JPanel rootPanel;

private JTable table;

private JButton deleteTableButton;

private JButton restoreButton;

private JButton saveTextButton;

private JButton loadingTextButton;

private JButton saveBinaryButton;

private JButton loadingBinaryButton;

private DefaultTableModel model;

private List<RecIntegral> data = new ArrayList<>();

private UDPThread server;

public App() throws SocketException {

server = new UDPThread();

server.start();

setVisible(true);

setSize(800, 600);

setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);

setContentPane(rootPanel);

model = (DefaultTableModel) table.getModel();

model.addColumn("up");

model.addColumn("down");

model.addColumn("step");

model.addColumn("result");

addButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String up = App.this.up.getText();

String down = App.this.down.getText();

String step = App.this.step.getText();

try {

if ((Float.*parseFloat*(up) < 0.000001) || (Float.*parseFloat*(up) > 1000000) || (Float.*parseFloat*(

down) < 0.000001) || (Float.*parseFloat*(down) > 1000000) || (Float.*parseFloat*(

step) < 0.000001) || (Float.*parseFloat*(step) > 1000000)) {

throw new ExceptionInput("wrong input");

}

} catch (ExceptionInput ex) {

App.this.up.setText("");

App.this.down.setText("");

App.this.step.setText("");

new ExceptionWarning();

return;

}

try {

if (Float.*parseFloat*(up) < Float.*parseFloat*(down)) {

throw new ExceptionInput("wrong input");

}

} catch (ExceptionInput ex) {

App.this.up.setText("");

App.this.down.setText("");

App.this.step.setText("");

new ExceptionUpDown();

return;

}

model.addRow(new String[]{up, down, step, "0"});

App.this.up.setText("");

App.this.down.setText("");

App.this.step.setText("");

data.add(new RecIntegral(Arrays.*stream*(new String[]{up, down, step, "0"}).toList()));

}

});

deleteButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int temp = table.getSelectedRow();

if (temp != -1) {

model.removeRow(temp);

data.remove(temp);

}

}

});

calculateButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int temp = table.getSelectedRow();

if (temp == -1) {

return;

}

Vector localData = model.getDataVector().get(temp);

float max = Float.*parseFloat*((String) localData.get(0));

float min = Float.*parseFloat*((String) localData.get(1));

float step = Float.*parseFloat*((String) localData.get(2));

if(server.getCountSockets() >= 3){

try {

server.sendMessages(min,max,step);

} catch (IOException ex) {

throw new RuntimeException(ex);

}

double result;

try {

result= server.getResults();

} catch (IOException ex) {

throw new RuntimeException(ex);

}

model.setValueAt(result,temp, 3);

data.get(temp).setDataByIndex(3, String.*valueOf*(result));

}

}

});

restoreButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int temp = table.getRowCount();

for (int i = 0; i < temp; i++) {

model.removeRow(0);

}

for (RecIntegral element : data) {

model.addRow(element.getRecord().toArray());

}

}

});

deleteTableButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

int temp = table.getRowCount();

for (int i = 0; i < temp; i++) {

model.removeRow(0);

}

}

});

model.addTableModelListener(new TableModelListener() {

@Override

public void tableChanged(TableModelEvent e) {

if (e.getType() == TableModelEvent.*UPDATE*) {

data.get(table.getSelectedRow()).setDataByIndex(table.getSelectedColumn(),

(String) model.getDataVector().get(table.getSelectedRow()).get(table.getSelectedColumn()));

}

}

});

saveBinaryButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

ObjectOutputStream out = null;

int temp = table.getSelectedRow();

if (temp == -1) {

return;

}

try {

out = new ObjectOutputStream(new BufferedOutputStream(

new FileOutputStream("BinaryStringNumber" + temp + ".txt")));

out.writeObject(data.get(temp));

out.close();

} catch (IOException ignored) {

}

}

});

loadingBinaryButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser(FileSystemView.*getFileSystemView*().getHomeDirectory());

fileChooser.showOpenDialog(null);

ObjectInputStream in = null;

RecIntegral restObj = null;

int temp = table.getRowCount();

if (temp != -1) {

for (int i = 0; i < temp; i++) {

model.removeRow(0);

}

}

try {

in = new ObjectInputStream(new BufferedInputStream(

new FileInputStream(fileChooser.getSelectedFile().getAbsolutePath())));

restObj = (RecIntegral) in.readObject();

data.add(restObj);

model.addRow(restObj.getRecord().toArray());

} catch (IOException | ClassNotFoundException ex) {

ex.printStackTrace();

}

}

});

saveTextButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

ObjectOutputStream out = null;

int temp = table.getSelectedRow();

if (temp == -1) {

return;

}

try {

out = new ObjectOutputStream(new BufferedOutputStream(

new FileOutputStream("TextStringNumber" + temp + ".txt")));

out.writeObject(data.get(temp).toString());

out.close();

} catch (IOException ignored) {

}

}

});

loadingTextButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

JFileChooser fileChooser = new JFileChooser(FileSystemView.*getFileSystemView*().getDefaultDirectory());

fileChooser.showOpenDialog(null);

ObjectInputStream in = null;

RecIntegral restObj = null;

int temp = table.getRowCount();

if (temp != -1) {

for (int i = 0; i < temp; i++) {

model.removeRow(0);

}

}

try {

in = new ObjectInputStream(new BufferedInputStream(

new FileInputStream(fileChooser.getSelectedFile().getAbsolutePath())));

restObj = RecIntegral.*fromString*((String) in.readObject());

data.add(restObj);

model.addRow(restObj.getRecord().toArray());

} catch (IOException | ClassNotFoundException ex) {

ex.printStackTrace();

}

}

});

}

public static void main(String[] args) throws SocketException {

new App();

}

private void createUIComponents() {

table = new JTable() {

@Override

public boolean isCellEditable(int row, int column) {

return column != 3;

}

};

}

}

UDPThread.java

package main.java;

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.util.ArrayList;

import java.util.List;

public class UDPThread extends Thread {

private int countSockets;

private int lastSending;

private List<byte[]> receivingDataBuffer;

private List<byte[]> sendingDataBuffer;

private List<DatagramPacket> input;

private List<DatagramPacket> output;

private DatagramSocket serverSocket;

private int port = 8080;

public UDPThread() throws SocketException {

serverSocket = new DatagramSocket(port);

countSockets = 0;

}

public void run() {

receivingDataBuffer = new ArrayList<>();

input = new ArrayList<>();

for (int i = 0; i < 3; i++) {

receivingDataBuffer.add(new byte[1024]);

input.add(new DatagramPacket(receivingDataBuffer.get(i), receivingDataBuffer.get(i).length));

try {

serverSocket.receive(input.get(i));

countSockets++;

} catch (IOException e) {

throw new RuntimeException(e);

}

}

}

public void sendMessages(double min, double max, double step) throws IOException {

lastSending = 0;

sendingDataBuffer = new ArrayList<>();

int lenModStep = (int) ((int) (max - Math.*abs*(min)) % step);

int stepsCount = (int) ((int) (max - Math.*abs*(min)) / step);

int threadCount = (lenModStep == 0 ? Math.*min*(stepsCount, 3) : stepsCount + 1 > 3 ? 3 : stepsCount);

if (threadCount == 3) {

int steps = stepsCount / 3;

sendingDataBuffer.add((min + " " + (min + steps \* step) + " " + step).getBytes());

sendingDataBuffer.add(((min + steps \* step) + " " + (min + steps \* 2 \* step) + " " + step).getBytes());

sendingDataBuffer.add(((min + steps \* 2 \* step) + " " + max + " " + step).getBytes());

for (int i = 0; i < 3; i++) {

InetAddress senderAddress = input.get(i).getAddress();

int senderPort = input.get(i).getPort();

DatagramPacket outputPacket = new DatagramPacket(

sendingDataBuffer.get(i), sendingDataBuffer.get(i).length,

senderAddress, senderPort);

serverSocket.send(outputPacket);

}

lastSending = 3;

} else if (threadCount == 2) {

int steps = stepsCount / 2;

sendingDataBuffer.add((min + " " + steps \* step + " " + step).getBytes());

sendingDataBuffer.add((min + steps \* step + " " + max + " " + step).getBytes());

for (int i = 0; i < 2; i++) {

InetAddress senderAddress = input.get(i).getAddress();

int senderPort = input.get(i).getPort();

DatagramPacket outputPacket = new DatagramPacket(

sendingDataBuffer.get(i), sendingDataBuffer.get(i).length,

senderAddress, senderPort);

serverSocket.send(outputPacket);

}

lastSending = 2;

} else {

sendingDataBuffer.add((min + " " + max + " " + step).getBytes());

InetAddress senderAddress = input.get(0).getAddress();

int senderPort = input.get(0).getPort();

DatagramPacket outputPacket = new DatagramPacket(

sendingDataBuffer.get(0), sendingDataBuffer.get(0).length,

senderAddress, senderPort);

serverSocket.send(outputPacket);

lastSending = 1;

}

}

public Double getResults() throws IOException {

double result = 0.0;

for (int i = 0; i< lastSending; i++){

input.get(i).setData(new byte[100]);

serverSocket.receive(input.get(i));

result += Double.*parseDouble*(new String(input.get(i).getData()));

}

return result;

}

public int getCountSockets() {

return countSockets;

}

public void setCountSockets(int countSockets) {

this.countSockets = countSockets;

}

}

Клиент

main.java

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class main {

private static final int *port* = 8080;

public static void main(String[] args) throws IOException {

try{

DatagramSocket clientSocket = new DatagramSocket();

InetAddress IPAddress = InetAddress.*getByName*("localhost");

byte[] sendingDataBuffer = new byte[1024];

byte[] receivingDataBuffer = new byte[1024];

String sentence = "Hello from UDP client";

sendingDataBuffer = sentence.getBytes();

DatagramPacket sendingPacket = new DatagramPacket(sendingDataBuffer,sendingDataBuffer.length,IPAddress, *port*);

clientSocket.send(sendingPacket);

DatagramPacket receivingPacket = new DatagramPacket(receivingDataBuffer,receivingDataBuffer.length);

clientSocket.receive(receivingPacket);

Double[] request = Arrays.*stream*(new String(receivingPacket.getData()).split(" ")).map(Double::*parseDouble*).toArray(Double[]::new);

double min = request[0];

double max = request[1];

double step = request[2];

int lenModStep = (int) ((int) (max - Math.*abs*(min)) % step);

int stepsCount = (int) ((int) (max - Math.*abs*(min)) / step);

int threadCount = (lenModStep == 0 ? Math.*min*(stepsCount, 3) : stepsCount + 1 > 3 ? 3 : stepsCount);

ResultResource rs = new ResultResource();

List<CalculationThread> threads = new ArrayList<>();

if (threadCount == 3) {

int steps = stepsCount / 3;

threads.add(new CalculationThread(min, min + steps \* step, step, rs));

threads.add(

new CalculationThread(min + steps \* step, min + steps \* 2 \* step, step,

rs));

threads.add(new CalculationThread(min + steps \* 2 \* step, max, step, rs));

} else if (threadCount == 2) {

int steps = stepsCount / 2;

threads.add(new CalculationThread(min, min + steps \* step, step, rs));

threads.add(new CalculationThread(min + steps \* step, max, step, rs));

} else {

threads.add(new CalculationThread(min, max, step, rs));

}

threads.forEach(CalculationThread::start);

while (rs.getChangeCount() != threadCount){

Thread.*sleep*(100);

}

sendingDataBuffer = String.*valueOf*(rs.getResult()).getBytes();

sendingPacket = new DatagramPacket(sendingDataBuffer,sendingDataBuffer.length,IPAddress, *port*);

clientSocket.send(sendingPacket);

clientSocket.close();

}

catch(SocketException e) {

e.printStackTrace();

} catch (InterruptedException e) {

throw new RuntimeException(e);

}

}

}

CalculationThread.java

public class CalculationThread extends Thread {

private double min;

private double max;

private double step;

private final ResultResource result;

public CalculationThread(double min, double max, double step, ResultResource result) {

this.min = min;

this.max = max;

this.step = step;

this.result = result;

}

public void run() {

double localResult = 0;

for (double i = min; i < max - step; i += step) {

if (i > max) {

localResult += (Math.*cos*(i - step) + Math.*cos*(max)) / 2 \* step;

} else {

localResult += (Math.*cos*(i) + Math.*cos*(i + step)) / 2 \* step;

}

}

synchronized (result) {

result.setChangeCount(result.getChangeCount() + 1);

result.setResult(result.getResult() + localResult);

}

}

}

Вывод: в ходе работы мы научились создавать клиент-серверные приложения c использованием стандартных классов Java.