Пензенский государственный университет

Факультет вычислительной техники

Кафедра «Вычислительная техника»

Сетевое взаимодействие в Java

Отчет о лабораторной работе № 6

по дисциплине «Разработка кроссплатформенных приложений»

Вариант № 2

Выполнили: ст-ты гр. 21ВОЭ1

Кокорин А. А.

Тишаев М. Р.

Приняли: к.т.н., доцент каф. ВТ

Юрова О. В.

к.т.н., доцент каф. ВТ

Карамышева Н. С.

2024

1 Цель работы

Научиться создавать клиент-серверные приложения c использованием стандартных классов Java.

2 Формулировка задания

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. Использовать TCP.

3 Описание программы

Программа построена на основе лабораторной работы № 5.

Текст программы приведен в приложении А.

Программа представляет собой серверное приложение с графическим интерфейсом (GUI) для вычисления интегралов с использованием многопоточности. Программа использует библиотеку Swing для создания GUI для работы с сетевыми соединениями.

Классы RecIntegral и ThreadIntegral:

RecIntegral: Класс, представляющий диапазон интегрирования и шаг (h).

ThreadIntegral: Класс, наследующий Thread, для вычисления интеграла на заданном участке.

Класс ServerGUI:

Основной класс, отвечающий за создание графического интерфейса и запуск сервера для распределения задач между клиентами.

Включает текстовые поля для ввода начала и конца диапазона интегрирования, количества частей и кнопку для запуска сервера.

Методы ServerGUI:

startServer(): Запускает серверный сокет, принимает подключения клиентов и распределяет задачи по вычислению интегралов.

distributeTasks(): Разбивает задачу на части, создаёт и запускает потоки для вычисления интегралов на поддиапазонах, собирает и выводит результаты.

Общий ход выполнения программы:

Пользователь вводит начало, конец диапазона интеграции и количество частей в соответствующие текстовые поля и нажимает кнопку "Start".

Запускается серверный сокет, который начинает ожидать подключений от клиентов.

После подключения нужного количества клиентов сервер распределяет задачи по вычислению интегралов между потоками.

Каждый поток вычисляет интеграл на своём поддиапазоне и возвращает результат.

Результаты всех потоков суммируются и выводятся

Класс Client:

В методе main устанавливается соединение с сервером, читается задача, выполняются вычисления и отправляется результат обратно на сервер.

Пошаговое описание работы программы

Установка соединения с сервером:

Socket socket = new Socket("localhost", 9999); - создаётся сокет для подключения к серверу на локальном хосте (localhost) и порту 9999.

Создание потоков для ввода и вывода:

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream())); - создаётся поток для чтения данных, получаемых от сервера.

PrintWriter out = new PrintWriter(socket.getOutputStream(), true); - создаётся поток для отправки данных на сервер.

Чтение задачи от сервера:

String task = in.readLine(); - клиент читает строку с задачей от сервера.

String[] parts = task.split(" "); - задача разбивается на части (начало, конец диапазона и шаг).

Выполнение вычислений:

double result = integrate(start, end, step); - вызывается метод integrate, который вычисляет интеграл на заданном диапазоне.

Отправка результата серверу:

out.println(result); - результат вычислений отправляется обратно на сервер.

Закрытие соединения:

socket.close(); - клиент закрывает сокет, завершая соединение с сервером.

Клиентское приложение подключается к серверу, получает задачу по вычислению интеграла, выполняет численное интегрирование методом трапеций и отправляет результат обратно на сервер.

4 Программа и методика испытаний

Для проверки правильности работы программы подготовлен вручную тестовый набор данных в таблице 1.

Таблица 1 – Тестовый набор данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a | b | h | Результат работы программы | Результат ручного просчета |
| 2 | 4 | 0,2 | 0,1047 | 0,1047 |
| 1 | 4 | 0,3 | 0,1526 | 0,1526 |
| 100 | 100000 | 0,01 | -19,9084 | -19,9084 |
| 24 | 200 | 0,02 | 106,2013 | 106,2013 |

При работе программы с тестовыми данными получены результаты, приведенные в Приложении Б. Результаты, приведенные в таблице 1, совпадают с результатами работы программы на рисунке Б.1. Таким образом, можно сделать вывод, что программа работает правильно.

Вывод

Результатом выполнения данной лабораторной работы является изучение создания клиент-серверного приложения c использованием стандартных классов Java.

ТЕКСТ ПРОГРАММЫ

Приложение А

(обязательное)

package my.javaapplication1;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

import java.net.\*;

import java.util.ArrayList;

import java.util.List;

class RecIntegral {

private double a;

private double b;

private double h;

public RecIntegral(double a, double b, double h) {

this.a = a;

this.b = b;

this.h = h;

}

public double getA() {

return a;

}

public double getB() {

return b;

}

public double getH() {

return h;

}

}

class ThreadIntegral extends Thread {

private double Res = 0;

private RecIntegral RecInt;

ThreadIntegral(String name, RecIntegral RecInt) {

super(name);

this.RecInt = RecInt;

}

public double getRes() {

return this.Res;

}

void findIntegral(double a, double b, double h) {

for (double i = a; i < b; i += h) {

this.Res += ((Math.sin(i \* Math.PI / 180) + Math.sin((i + h) \* Math.PI / 180)) \* (h / 2));

}

}

public void run() {

findIntegral(RecInt.getA(), RecInt.getB(), RecInt.getH());

}

}

public class ServerGUI extends JFrame {

private JTextField startField;

private JTextField endField;

private JTextField partsField;

private JTextArea logArea;

private JButton startButton;

private ServerSocket serverSocket;

private List<Socket> clientSockets;

private int clientCount = 0;

public ServerGUI() {

setTitle("Integration Server");

setSize(500, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

initUI();

}

private void initUI() {

JPanel panel = new JPanel(new GridLayout(4, 2));

panel.add(new JLabel("Start:"));

startField = new JTextField("");

panel.add(startField);

panel.add(new JLabel("End:"));

endField = new JTextField("");

panel.add(endField);

panel.add(new JLabel("Parts:"));

partsField = new JTextField("");

panel.add(partsField);

startButton = new JButton("Start");

startButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

startServer();

}

});

panel.add(startButton);

logArea = new JTextArea();

logArea.setEditable(false);

add(panel, BorderLayout.NORTH);

add(new JScrollPane(logArea), BorderLayout.CENTER);

}

private void startServer() {

try {

int parts = Integer.parseInt(partsField.getText());

double start = Double.parseDouble(startField.getText());

double end = Double.parseDouble(endField.getText());

double interval = (end - start) / parts;

serverSocket = new ServerSocket(9999);

clientSockets = new ArrayList<>();

logArea.append("Server started. Waiting for clients...\n");

new Thread(() -> {

try {

while (clientCount < parts) {

Socket clientSocket = serverSocket.accept();

clientSockets.add(clientSocket);

logArea.append("Client connected: " + clientSocket + "\n");

clientCount++;

}

distributeTasks(start, interval, parts);

} catch (IOException ex) {

ex.printStackTrace();

}

}).start();

} catch (IOException ex) {

ex.printStackTrace();

}

}

private void distributeTasks(double start, double interval, int parts) {

try {

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

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

double subStart = start + i \* interval;

double subEnd = subStart + interval;

double step = (subEnd - subStart) / 1000;

RecIntegral recInt = new RecIntegral(subStart, subEnd, step);

ThreadIntegral thread = new ThreadIntegral("Thread-" + (i + 1), recInt);

threads.add(thread);

thread.start();

}

for (ThreadIntegral thread : threads) {

thread.join();

logArea.append(thread.getName() + " result: " + thread.getRes() + "\n");

}

double totalResult = threads.stream().mapToDouble(ThreadIntegral::getRes).sum();

logArea.append("Total Result: " + totalResult + "\n");

} catch (InterruptedException ex) {

ex.printStackTrace();

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

@Override

public void run() {

ServerGUI server = new ServerGUI();

server.setVisible(true);

}

});

}

}

package my.javaapplication1;

import java.io.\*;

import java.net.\*;

public class Client {

public static void main(String[] args) {

try {

Socket socket = new Socket("localhost", 9999);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

String task = in.readLine();

String[] parts = task.split(" ");

double start = Double.parseDouble(parts[0]);

double end = Double.parseDouble(parts[1]);

double step = Double.parseDouble(parts[2]);

double result = integrate(start, end, step);

out.println(result);

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

private static double integrate(double start, double end, double step) {

double sum = 0.0;

for (double x = start; x < end; x += step) {

sum += ((Math.sin(x \* Math.PI / 180) + Math.sin((x + step) \* Math.PI / 180)) \* (step / 2));

}

return sum;

}

}

РЕЗУЛЬТАТЫ ИСПЫТАНИЙ

Приложение Б

(обязательное)

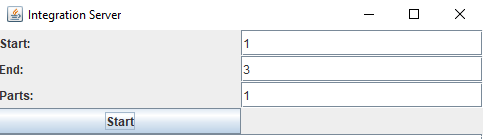


Рисунок Б.1 – Вычисление определенного интеграла

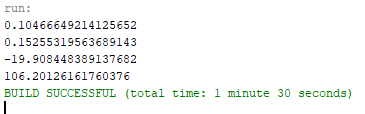


Рисунок Б.2 – Результаты выполнения программы Клиента

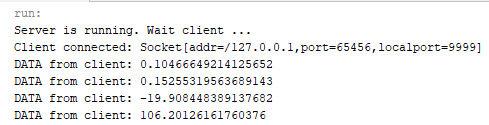


Рисунок Б.3 – Результаты выполнения программы Сервера