Министерство науки и высшего образования РФ

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Кафедра Вычислительной техники

Новосибирск 2020

|  |  |
| --- | --- |
| Факультет: АВТФ  Группа: АВТ-814  Студент: Альжанов В.А. | Преподаватель:  Васюткина Ирина Александровна |

Лабораторная работа 6

«Сетевые «клиент-серверные» приложения»

по дисциплине «Технологии программирования»

**Цель работы**

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

**Задание**

Вариант 2

1. создать отдельное консольное приложение, которое будет играть роль TCP-сервера. Клиентом будет приложение, которое было создано в предыдущих работах;
2. 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;
3. запрограммировать специальное взаимодействие по TCP с другими клиентами через сервер по варианту;

Реализовать возможность отправить N случайных объектов из текущей симуляции другому подключенному клиенту.

**Демонстрация работы программы**

////...

**Описание блоков программы**

class Server – класс, реализующий TCP-сервер. Содержит поля TreeMap<Integer, ConnectionThread> clients (хранение всех потоков обработки клиентов), List<Integer> ids (хранение идентификаторов клиентов), а также метод main(String[] args), позволяющий выполнять работу сервера в отдельном приложении. В методе создается ServerSocket с портом 1337 и начинается его прослушивание на установление подключения. При обнаружении подключения создается экземпляр класса ConnectionThread, который отвечает за общение с подключенным клиентом. После этого производится отправка обновленных данных о подключенных клиентах.

public class Server {  
 public static final String *SWAP\_WITH\_CLIENT* = "connect client";  
 public static final String *DISCONNECT* = "disconnect";  
  
 static TreeMap<Integer, ConnectionThread> *clients* = new TreeMap<>();  
 static List<Integer> *ids* = new ArrayList<>();  
 private static int *id* = 0;  
  
 public static void main(String[] args) {  
 try {  
 System.*out*.println("server.Server is running");  
 int port = 1337;  
 ServerSocket serverSocket = new ServerSocket(port);  
 new Ssd().start();  
 // Ждет клиентов и для каждого создает отдельный поток  
  
 while (true) {  
 Socket s = serverSocket.accept();  
 ConnectionThread connectionThread = new ConnectionThread(s, *id*);  
 *clients*.put(*id*, connectionThread);  
 *ids*.add(*id*);  
 *id*++;  
 connectionThread.start();  
 System.*out*.println(*clients*);  
 *update*();  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 static void update(){  
 for (int i = 0; i < *ids*.size()-1; i++) {  
 Server.*clients*.get(*ids*.get(i)).updateClients();  
  
 }  
 }  
 static void updateAll(){  
 for (int i = 0; i < *ids*.size(); i++) {  
 Server.*clients*.get(*ids*.get(i)).updateClients();  
 }  
 }  
}

class Ssd – класс, предназначенный для ожидания клиента и создания для каждого отдельного потока.

class Ssd extends Thread {  
 public void run() {  
 while (true) {  
 System.*out*.println(Server.*clients*);  
 try {  
 *sleep*(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

class ConnectionThread – класс, отвечающий за общение с клиентом. Унаследован от класса Thread. Содержит поля ObjectOutputStream outStream (поток вывода в клиент), ObjectInputStream inStream (поток ввода с клиента), метод diconnect() (разрыв соединения с клиентом), метод updateClients() (отсылка клиента списка других подключенных клиентов), а также переопределенный метод run(), в котором производится сначала сериализация потоков ввода-вывода, затем передача подключенных клиентов, далее происходит ожидание команды от клиента, а при поступлении команды полученные данные сравниваются с набором доступных команд.

class ConnectionThread extends Thread {  
 private ObjectOutputStream outStream;  
 private ObjectInputStream inStream;  
 private Socket socket;  
 private int id;  
 private boolean connected = true;  
  
 ConnectionThread(Socket socket, int id) {  
 this.socket = socket;  
 this.id = id;  
 }  
  
 @Override  
 public void run() {  
 try {  
 outStream = new ObjectOutputStream(socket.getOutputStream());  
 inStream = new ObjectInputStream(socket.getInputStream());  
 System.*out*.println("Запущен поток для клиента " + id);  
  
 outStream.writeObject(Server.*ids*);  
 outStream.writeInt(id);  
 outStream.flush();  
  
 while (connected) {  
 System.*out*.println("read");  
 String command = inStream.readUTF();  
 System.*out*.println(command);  
 switch (command) {  
 case Server.*SWAP\_WITH\_CLIENT*:  
 int targetID = inStream.readInt();  
 List<Bee> Drones = (ArrayList<Bee>) inStream.readObject();  
 Server.*clients*.get(targetID).outStream.writeUTF("getBees");  
 Server.*clients*.get(targetID).outStream.writeObject(Drones);  
  
 Server.*clients*.get(targetID).outStream.flush();  
 break;  
  
 case Server.*DISCONNECT*:  
  
 connected = false;  
 disconnect();  
 break;  
 default:  
 System.*err*.println("Unknown command");  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 disconnect();  
 Server.*updateAll*();  
 }  
 }  
  
 private void disconnect() {  
 try {  
 System.*out*.println("disconnect");  
 if (inStream != null) inStream.close();  
 if (outStream != null) outStream.close();  
 socket.close();  
 connected = false;  
 Server.*clients*.remove(id);  
 for (int i = 0; i < Server.*ids*.size(); i++)  
 if (Server.*ids*.get(i) == id)  
 Server.*ids*.remove(i);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void updateClients() {  
 try {  
 outStream.writeUTF("updateClients");  
 List<Integer> isds = new ArrayList<>();  
 for (int i = 0; i < Server.*ids*.size(); i++)  
 isds.add(Server.*ids*.get(i));  
 System.*out*.println(isds);  
 outStream.writeObject(isds);  
 outStream.flush();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
  
 }  
}

class Client – класс клиента, унаследованный от класса Thread. Содержит в себе поля OjectOutput outStream (поток вывода данных на сервер), ObjectInputStream inStream (поток ввода с сервера), целочисленные поля targetID (номер клиента, с которым производится обмен), id (собственный номер клиента), а также переопределенный метод run(), в котором сначала происходит инициализация потоков ввода-вывода, затем получение подключенных клиентов, после чего происходит ожидание команд от сервера и отсылка собственных команд на сервер, при поступлении данных они сравниваются с набором доступных команд.

public class Client extends Thread {  
  
 private Socket socket;  
 private ObjectOutputStream outStream;  
 private ObjectInputStream inStream;  
 private boolean connected = true;  
 private String command;  
  
 private NetworkDialog owner;  
 private int targetId;  
 private int id;  
  
 Client(NetworkDialog owner) {  
 this.owner = owner;  
 }  
  
 @Override  
 public void run() {  
 try {  
 String host = "localhost";  
 int port = 1337;  
 socket = new Socket(host, port);  
 connected = true;  
 System.*out*.println("Соединение установлено");  
  
 outStream = new ObjectOutputStream(socket.getOutputStream());  
 inStream = new ObjectInputStream(socket.getInputStream());  
 owner.showText("Соединение установлено");  
  
 List<Integer> ids = (List<Integer>) inStream.readObject();  
 id = inStream.readInt();  
 owner.showList(ids, id);  
  
 while (connected) {  
 if (inStream.available() == 0) {  
 if (command != null) {  
 switch (command) {  
 case "get":  
 outStream.writeUTF(Server.*SWAP\_WITH\_CLIENT*);  
 outStream.writeInt(targetId);  
 outStream.writeObject(Habitat.*getBee*());  
 outStream.flush();  
  
 command = null;  
 break;  
 case "disconnect":  
 System.*out*.println("disc");  
 outStream.writeUTF(Server.*DISCONNECT*);  
 outStream.flush();  
 command = null;  
 break;  
 default:  
 command = null;  
 break;  
 }  
 }  
 } else {  
 String serverCommand = inStream.readUTF();  
 System.*out*.println(serverCommand);  
  
 switch (serverCommand) {  
 case "getBees":  
  
 ArrayList<Bee> bees = (ArrayList<Bee>) inStream.readObject();  
  
 for (int i = 0; i < bees.size(); i++) {  
 Habitat.*array*.add(bees.get(i));  
  
 }  
 break;  
  
 case "updateClients":  
 ids = (List<Integer>) inStream.readObject();  
 System.*out*.println("ids");  
 System.*out*.println(ids);  
 owner.showList(ids, id);  
  
 break;  
 }  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 closeAll();  
 }  
 }  
  
 public void swapWithClient(int id) {  
 command = "get";  
 targetId = id;  
 }  
  
 public void disconnect() {  
 command = "disconnect";  
 owner.showText("Соединения нет");  
 }  
  
 private void closeAll() {  
 System.*out*.println("closeAll");  
 connected = false;  
 try {  
 inStream.close();  
 outStream.close();  
 socket.close();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

class NetworkDialog – класс-наследник класса JDialog, предназначен для установления соединения и отображения списка клиентов.

public class NetworkDialog extends JDialog {  
  
 private JLabel statusLabel;  
 private JList<String> clients;  
 private Client client;  
 private String[] items;  
 private List<Integer> clientList;  
  
 NetworkDialog(GUI owner) {  
 super(owner, "Текущие объекты", false);  
 setLayout(new BorderLayout());  
 setBounds(700, 380, 400, 250);  
  
 JPanel clientsPanel = new JPanel(new BorderLayout());  
 clientsPanel.setBorder(new EmptyBorder(10, 15, 10, 10));  
 add(clientsPanel, BorderLayout.*CENTER*);  
  
 clients = new JList<>();  
 clientsPanel.add(clients);  
  
 JPanel buttons = new JPanel(new FlowLayout(FlowLayout.*LEFT*));  
 buttons.setPreferredSize(new Dimension(150, 170));  
 buttons.setBorder(new EmptyBorder(10, 10, 10, 15));  
 buttons.setAlignmentX(Component.*LEFT\_ALIGNMENT*);  
 add(buttons, BorderLayout.*WEST*);  
  
 JButton connectButton = new JButton("Подключится");  
 connectButton.addActionListener(e -> connect());  
 connectButton.setPreferredSize(new Dimension(125, 25));  
 connectButton.setFocusable(false);  
 buttons.add(connectButton);  
  
 JButton disconnectButton = new JButton("Отключится");  
 disconnectButton.addActionListener(e -> disconnect());  
 disconnectButton.setFocusable(false);  
 disconnectButton.setPreferredSize(connectButton.getPreferredSize());  
 buttons.add(disconnectButton);  
  
 JButton getButton = new JButton("Передать");  
 getButton.addActionListener(e -> get());  
 getButton.setPreferredSize(connectButton.getPreferredSize());  
 getButton.setFocusable(false);  
 buttons.add(getButton);  
  
  
  
 statusLabel = new JLabel("Соединения нет");  
 buttons.add(statusLabel);  
  
 setVisible(false);  
 }  
  
 private void get() {  
 if (clientList == null || clientList.isEmpty()) return;  
 if (clients.getSelectedIndex() == -1) return;  
  
 client.swapWithClient(clientList.get(clients.getSelectedIndex()));  
 }  
  
 public void showText(String text) {  
 statusLabel.setText(text);  
 }  
  
 private void connect() {  
 client = new Client(this);  
 client.start();  
 }  
  
 private void disconnect() {  
 client.disconnect();  
 clients.setListData(new String[0]);  
  
 }  
  
 public void showList(List<Integer> ids,int id) {  
 clientList = ids;  
 items = null;  
 items = new String[ids.size()];  
 for (int i = 0; i < ids.size(); i++) {  
 if (ids.get(i) == id){  
 items[i] = "Клиент №" + ids.get(i) + " <- Это вы";  
 }else{  
 items[i] = "Клиент №" + ids.get(i);  
 }  
  
 System.*out*.println(items[i]);  
 }  
 clients.setListData(items);  
 }  
}

В класс Habitat были добавлен статический метод getBee позволяющий передающий объектры из одного клиента другому.

static ArrayList<Bee> getBee(){  
 ArrayList<Bee> bee = new ArrayList<>();  
  
 for (int i = 0; i <*N* &&i<*array*.size(); i++) {  
  
 bee.add(*array*.get(0));  
 if (*array*.get(0) instanceof Drone) {  
 *numberOfDrones*--;  
 *iK*--;  
 }  
 else *numberOfWorkers*--;  
  
 *array*.remove(0);  
 }  
 return bee;  
}

В меню «Управление» был добавлен пункт «Сервер», при нажатии на который происходит показ диалогового окна NetworkDialog.

JMenuItem networkItem = new JMenuItem("Сервер");  
networkItem.addActionListener(e -> {  
 networkDialog = new NetworkDialog(this);  
 networkDialog.setVisible(true);  
});  
menuSimulation.add(networkItem);

**Выводы**

В ходе выполнения работы были изучены особенности реализации сетевых «клиент-серверных» приложений в Java.