МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра вычислительной техники

**Отчёт**

**по лабораторной работе №3**

**по дисциплине «Разработка клиент-серверных приложений»**

Тема: Сетевое соединение на сокетах

Группа: АВТ-909

Студенты: Буйко К.А. Пилецкий Д.К.

Преподаватель: Токарев В.Г.

НОВОСИБИРСК

2022

**Цель работы**

Изучить способы установления соединения между клиентом и сервером и передачи данных между ними.

**Задание**

Серверная часть реализуется в потоке и запускается/останавливается при помощи флажка. Серверная компонента ждет запроса на соединение на фиксированном порте. Клиентская компонента в другой программе запрашивает соединение с серверной. Протокол является симметричным (двунаправленным), т.е. каждое приложение может посылать один тот же набор команд: закрытие соединения, очистка вектора объектов, передача сериализованного объекта, запрос размера вектора объектов, запрос объекта с заданным номером из вектора. Принимаемые объекты отображаются в клиенте и запоминаются в векторе.

**Ход выполнения работы**

В лабораторной работе дополнялись ранее разработанные классы объектов: SocketServer (основанный на ServerSocket), SocketClient (основанный на Socket).

SocketClient является вспомогательным классом для уже разработанной программы. Выполняет роль связи с сервером и отправки команд о состоянии и данных об объектах.

SocketServer реализует консольное приложение для принятия команд и объектов от клиента.

Для начала работы необходимо установить соединения с сервером по указанному адресу и порту.

В обоих классах реализован базовый функционал для открытия потоков ввода и вывода и обработки сообщений, проходящих через них.

Клавиши для управления сервером через клиента:

“ p ” – открыть соединение, запустить сокет;

Создается экземпляр класса клиентского соединения.

На сервер отправляется команда «start».

На сервере обрабатывается данная команда и выводится информация о подключении.

“ o ” – остановить соединение, закрыть сокет;

На сервер отправляется команда «stop».

На сервере обрабатывается данная команда и производится закрытие соединения.

“ y ” – отправить все экземпляры Car на сервер с перезаписью;

На сервер отправляется команда «add».

На сервере обрабатывается данная команда и ожидается передача объекта коллекции целевых объектов, сериализованных стандартным образом (см. документацию Serialize Java [https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html]). После чего полученная коллекция заменяет текущую.

“ ‘ ” – отправить случайную Car на сервер;

На сервер отправляется команда «addOne».

На сервере обрабатывается данная команда и ожидается передача целевого объекта, сериализованного стандартным образом. После чего полученный объект дописывается в конце текущей коллекции на сервере.

“ ; ” – получить с сервера все экземпляры Cars с перезаписью;

На сервер отправляется команда «getVec».

На сервере обрабатывается данная команда и отправляется коллекция. Клиент ожидает загрузки сериализованной коллекции, которая дополняет текущую на клиенте.

“ l ” – получить с сервера один экземпляр Car;

На сервер отправляется команда «getOne». После неё индекс желаемого объекта.

На сервере обрабатывается данная команда и отправляется объект с соответствующим индексом, если индекс некорректный, ничего не отправляется. Клиент ожидает загрузки сериализованного объекта, который дополняет текущую коллекцию на клиенте.

“ u ” – вывести на сервере размер имеющегося у него вектора;

На сервер отправляется команда «size».

На сервере обрабатывается данная команда и отправляется число, обозначающее размер коллекции на сервере.

“ i ” – очистить вектор Cars на стороне сервера.

На сервер отправляется команда «clear».

На сервере обрабатывается данная команда и очищается текущая коллекция на сервере.

Листинг двух классов представлен в приложении.

**Вывод**

В ходе выполнения лабораторной работы были практически закреплены навыки обмена сообщениями между клиентом и сервером.

Также была создана программа моделирования создания, удаления, движения, сохранения и загрузки различными методами объектов, а также передача их на сервер и получение с него.

Данная реализованная модель соединения имеет преимущества и недостатки. Преимуществом является то, что для соединения используется протокол TCP, гарантирующий доставку и целостность сообщений. Однако, минусом является необходимость создавать объект ServerSocket для каждого приложения и, соответственно сохранять открытую сессию с каждым Socket клиента, что занижает эффективность сервера при масштабировании.

ПРИЛОЖЕНИЕ

Листинг класса SocketServer:

package Server;  
  
import Main.Car;  
import Main.Cargo;  
import Main.PassCar;  
  
import java.io.\*;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.ArrayList;  
  
public class SocketServer {  
 public static ArrayList<Car> *cars* = new ArrayList<>();  
 public static final int *SERVER\_PORT* = 50001;  
  
 public static void main(String[] args) {  
 ServerSocket server;  
 try {  
 server = new ServerSocket(*SERVER\_PORT*);  
 } catch (IOException e) {  
 e.printStackTrace();  
 return;  
 }  
  
 while (true) {  
 try {  
 Socket clientConn = server.accept();  
 InputStream serverInput = clientConn.getInputStream();  
 BufferedReader br = new BufferedReader(new InputStreamReader(serverInput));  
  
 DataOutputStream serverOutput = new DataOutputStream(clientConn.getOutputStream());  
  
 while (!clientConn.isClosed()) {  
 String command = br.readLine();  
 System.*out*.println("Command on server: " + command);  
 if (command == null)  
 continue;  
 switch (command) {  
 case "stop": {  
 System.*out*.println("Client disconnected.");  
 clientConn.close();  
 break;  
 }  
 case "start": {  
 System.*out*.println("Client connected.");  
 break;  
 }  
 case "clear": {  
 System.*out*.println("Vector was cleared.");  
 *cars*.clear();  
 break;  
 }  
 case "size": {  
 System.*out*.println("return size: " + *cars*.size() + ".");  
 serverOutput.writeInt(*cars*.size());  
 serverOutput.flush();  
 break;  
 }  
 case "add": {  
 System.*out*.println("Add vector.");  
 System.*out*.println("Vector size before operation: " + *cars*.size() + ".");  
 ObjectInputStream f = new ObjectInputStream(new BufferedInputStream(serverInput));  
  
 try {  
 try {  
 *cars* = (ArrayList<Car>) f.readObject();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("Vector size after operation: " + *cars*.size() + ".");  
 break;  
 }  
 case "addOne": {  
 System.*out*.println("Add One to vector.");  
 System.*out*.println("Vector size before operation: " + *cars*.size() + ".");  
 ObjectInputStream f = new ObjectInputStream(new BufferedInputStream(serverInput));  
  
 try {  
 String name = f.readUTF();  
 Car c;  
 try {  
 Class<?> t = Class.*forName*("Main." + name);  
 if (name.equals("Cargo"))  
 c = (Cargo) t.newInstance();  
 else  
 c = (PassCar) t.newInstance();  
 c.Deserialize(f);  
 *cars*.add(c);  
 } catch (ClassNotFoundException | IllegalAccessException | InstantiationException e) {  
 e.printStackTrace();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 System.*out*.println("Vector size after operation: " + *cars*.size() + ".");  
 break;  
 }  
 case "getVec": {  
 try {  
 System.*out*.println("Send Vector.");  
 ObjectOutputStream f = new ObjectOutputStream(new BufferedOutputStream(serverOutput));  
 f.writeObject(*cars*);  
 f.flush();  
 System.*out*.println("OK.");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 break;  
 }  
 case "getOne": {  
 int index = br.read();  
 ObjectOutputStream f = new ObjectOutputStream(new BufferedOutputStream(serverOutput));  
 if (index < 0 || index > *cars*.size() - 1) {  
 f.writeObject(null);  
 } else {  
 f.writeObject(*cars*.get(index));  
 }  
 f.flush();  
 System.*out*.println("Object send.");  
 break;  
 }  
 default:  
 break;  
 }  
  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
}

Листинг класса SocketClient:

package Main;  
  
import java.io.\*;  
import java.net.Socket;  
import java.util.ArrayList;  
  
public class SocketClient {  
 static Socket *clientSocket* = null;  
 static DataOutputStream *os* = null;  
 static InputStream *is* = null;  
 static BufferedReader *br* = null;  
  
 public static void start() {  
 if (*clientSocket* == null) {  
 try {  
 *clientSocket* = new Socket("localhost", 50001);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 try {  
 *os* = new DataOutputStream(*clientSocket*.getOutputStream());  
 *os*.writeBytes("start\n");  
 *os*.flush();  
 *is* = *clientSocket*.getInputStream();  
 *br* = new BufferedReader(new InputStreamReader(*is*));   
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } else {  
 System.*out*.println("This client is already connected.");  
 }  
 }  
  
 public static void stop() {  
 System.*out*.println(1);  
 if (*clientSocket* != null) {  
 System.*out*.println(2);  
 try {  
 *os*.writeBytes("stop\n");  
 *os*.flush();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } else {  
 System.*out*.println("This client hasn't connection.");  
 }  
 *clientSocket* = null;  
 }  
  
 public static void clearVec() {  
 if (*clientSocket* != null) {  
 try {  
 *os*.writeBytes("clear\n");  
 *os*.flush();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } else {  
 System.*out*.println("This client hasn't connection.");  
 }  
 }  
  
 public static int getVecSize() {  
 if (*clientSocket* != null) {  
 try {  
 *os*.writeBytes("size\n");  
 *os*.flush();  
 int size = *br*.read();  
 return size;  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } else {  
 System.*out*.println("This client hasn't connection.");  
 }  
 return -1;  
 }  
  
 public static void addToServer(ArrayList<Car> cars, int index) {  
 if (*clientSocket* != null) {  
 try {  
 ObjectOutputStream f = new ObjectOutputStream(new BufferedOutputStream(*os*));  
 if (index == -2) {  
 *os*.writeBytes("add\n");  
 *os*.flush();  
 f.writeObject(cars);  
 f.flush();  
 } else {  
 if (0 <= index && index < cars.size()) {  
 *os*.writeBytes("addOne\n");  
 *os*.flush();  
 cars.get(index).Serialize(f);  
 f.flush();  
 }  
 }  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } else {  
 System.*out*.println("This client hasn't connection.");  
 }  
 }  
  
 public static ArrayList<Car> getVecFromServer() {  
 ArrayList<Car> cars = null;  
 if (*clientSocket* != null) {  
 try {  
 *os*.writeBytes("getVec" + '\n');  
 *os*.flush();  
 ObjectInputStream f = new ObjectInputStream(new BufferedInputStream(*is*));  
 cars = (ArrayList<Car>) f.readObject();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } else {  
 System.*out*.println("This client hasn't connection.");  
 }  
 return cars;  
 }  
  
 public static Car getOneFromServer(int index) {  
 Car car = null;  
 if (*clientSocket* != null) {  
 try {  
 *os*.writeBytes("getOne" + '\n');  
 *os*.write(index);  
 *os*.flush();  
 ObjectInputStream f = new ObjectInputStream(new BufferedInputStream(*is*));  
 car = (Car) f.readObject();  
 System.*out*.println("Object received.");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 } else {  
 System.*out*.println("This client hasn't connection.");  
 }  
 return car;  
 }  
}