|  |
| --- |
|  |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |
| Институт информационных технологий (ИТ) |
| Кафедра инструментального и прикладного программного обеспечения |

|  |  |
| --- | --- |
| **ОТЧЕТ**  **ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2** | |
| **по дисциплине** |  |
| **«Архитектура клиент-серверных приложений»**  **Тема: Программирование JAVA сокетов** | |
| Выполнил студент группы ИКБО-20-22 | Шумахер М.Е. |
| Принял преподаватель кафедры ИиППО | Волков М.Ю. |

|  |  |  |
| --- | --- | --- |
| Практические работы выполнены | «04» сентября 2024 г. |  |
| «Зачтено» | «04» сентября 2024 г. |  |

Москва 2024

**Цель работы:** знакомство студентов с одной из базовых технологии реализации конечной точки для передачи и приема данных по сети - сокетом.

**Теоретическое введение:**

Клиент-серверная архитектура - это самая известная структура приложений в Интернете. В этой архитектуре клиенты (например, персональные компьютеры, устройства IoT и т. Д.) вначале запрашивают ресурсы с сервера, а затем сервер отправляет соответствующие ответы на запросы клиентов. Чтобы это произошло, как на клиентах, так и на серверах должен быть реализован какой-то механизм, поддерживающий эту сетевую транзакцию. Этот механизм называется коммуникацией через сокеты.

Стоит отметить, что существует два типа сокетов для TCP и UDP. Поскольку большинство сетевых приложений используют TCP, в тексте работы речь пойдёт только о TCP-сокетах и их реализации. Основное различие между ними заключается в том, что UDP не имеет соединения, то есть между клиентом и сервером нет сеанса, в то время как TCP ориентирован на соединение, то есть сначала должно быть установлено эксклюзивное соединение между клиентом и сервером для связи.

Сокет работает по типичной модели запрос / ответ, где в java-программе, называемой клиентом, вызывается другая программа, называемая сервером, работающая на другой JVM. Клиент должен отправить запрос, и сервер отвечает ответом. В этой модели обмен всегда инициируется клиентом; сервер не может отправлять какие-либо данные без предварительного запроса клиента. Стоит отметить, что данная модель хорошо работала во всемирной паутине, когда клиенты время от времени запрашивали документы, которые нечасто менялись, но ограничения этого подхода становятся все более актуальными, поскольку контент меняется быстро и пользователи ожидают более интерактивного взаимодействия в сети. Например, WebSocket устраняет эти ограничения, предоставляя полнодуплексный канал связи между клиентом и сервером. Протокол WebSocket (JSR 356) будет рассмотрен в следующей практической работе. Но на практике, сокеты активно используются при реализации клент-серверного взаимодействия.

Сокет - это одна из конечных точек двустороннего канала связи между двумя программами, работающими в сети. Сокет привязан к номеру порта, чтобы уровень TCP мог идентифицировать приложение, данные предназначенные для отправки.

Каждый сервер - это программа, которая работает в определенной системе и прослушивает определенный порт. Сокеты привязаны к номерам портов, и когда мы запускаем любой сервер, он просто слушает сокет и ждет клиентских запросов. Например, сервер tomcat, работающий на порту 8080, ожидает клиентских запросов и, получив любой клиентский запрос, отвечает на них.

Следовательно, для запуска любого сокета необходим запускаемый порт. Очевидно, что он размещен на машине, идентифицируемой именем хоста и уникальной адресованной IP.

В Java классы для поддержки программирования сокетов упакованы в пакет java.net. В соответствии с Java, java.net пакет поставляется с двумя классами Socket и ServerSocket для функций клиента и сервера соответственно

**Постановка задачи:** 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.

**Программный код:**

import java.net.Socket;  
  
public class Client {  
 public static void main(String... args) {  
 try (Socket socket = new Socket("localhost", 8080)) {  
 System.*out*.println("Стартовал клиент!");  
 ClientReader reader = new ClientReader(socket);  
 ClientSender sender = new ClientSender(socket);  
  
 sender.start();  
 reader.start();  
  
 sender.join();  
 reader.join();  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
}

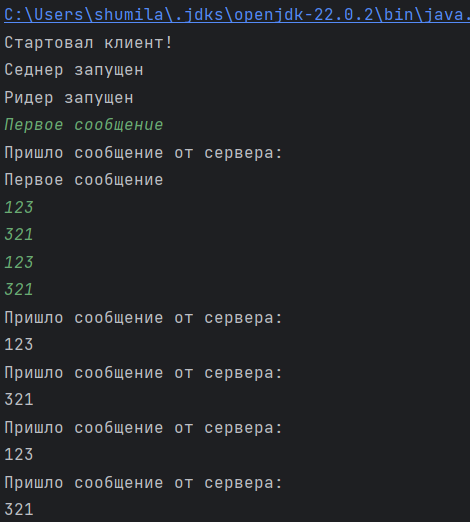
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.net.Socket;  
  
public class ClientReader extends Thread{  
 private Socket socket;  
 public ClientReader(Socket socket) {  
 this.socket = socket;  
 }  
  
 @Override  
 public void run() {  
 try {  
 System.*out*.println("Ридер запущен");  
 BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 String message;  
 while (true) {  
 message = in.readLine();  
 if (message != null) {  
 System.*out*.println("Пришло сообщение от сервера:\n" + message);  
 }  
 }  
 } catch (IOException e) {  
 System.*out*.println(e.getMessage());  
 }  
  
 }  
}

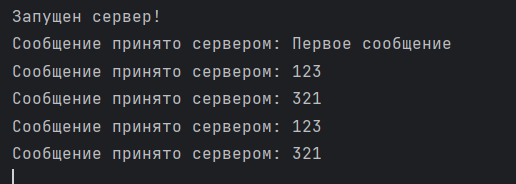
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.Socket;  
  
public class ClientSender extends Thread {  
 private Socket socket;  
 public ClientSender(Socket socket) {  
 this.socket = socket;  
 }  
  
 @Override  
 public void run() {  
  
 try {  
 System.*out*.println("Седнер запущен");  
 PrintWriter out = new PrintWriter(socket.getOutputStream(), true);  
 BufferedReader userInput = new BufferedReader(new InputStreamReader(System.*in*));  
 String input;  
 while ((input = userInput.readLine()) != null) {  
 out.println(input);  
 }  
 } catch (IOException e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
}

import java.io.BufferedReader;  
import java.io.InputStreamReader;  
import java.io.PrintWriter;  
import java.net.Socket;  
import java.util.List;  
  
public class ClientHandler implements Runnable {  
 private Socket socket;  
 private BufferedReader in;  
 private PrintWriter out;  
 private List<String> messageBuffer;  
 private List<ClientHandler> clients;  
  
 public ClientHandler(Socket socket, List<ClientHandler> clients, List<String> messageBuffer) {  
 this.socket = socket;  
 this.clients = clients;  
 this.messageBuffer = messageBuffer;  
 }  
  
 @Override  
 public void run() {  
 try {  
 in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
 out = new PrintWriter(socket.getOutputStream(), true);  
 String message;  
 while ((message = in.readLine()) != null) {  
 System.*out*.println("Сообщение принято сервером: " + message);  
 synchronized (messageBuffer) {  
 messageBuffer.add(message);  
 }  
 }  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 } finally {  
 try {  
 clients.remove(this);  
 socket.close();  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
 }  
  
 public void sendMessage(String msg) {  
 out.println(msg);  
 }  
}

import java.io.IOException;  
import java.net.ServerSocket;  
import java.net.Socket;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
  
public class Server {  
  
 private static final List<ClientHandler> *clients* = new ArrayList<>();  
 private static final List<String> *messages* = new ArrayList<>();  
  
 public static void main(String... args) throws IOException {  
 ExecutorService executorService = Executors.*newCachedThreadPool*();  
  
 try (ServerSocket serverSocket = new ServerSocket(8080)) {  
 System.*out*.println("Запущен сервер!");  
 executorService.execute(() -> {  
 while (true) {  
 try {  
 Thread.*sleep*(5000);  
 synchronized (*messages*) {  
 if (!*messages*.isEmpty()) {  
 *sendMessagesToClients*();  
 *messages*.clear();  
 }  
 }  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
 });  
 while (true) {  
 Socket socket = serverSocket.accept();  
 ClientHandler clientHandler = new ClientHandler(socket, *clients*, *messages*);  
 *clients*.add(clientHandler);  
 executorService.execute(clientHandler);  
 }  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
  
  
 private static void sendMessagesToClients() {  
 for (ClientHandler clientHandler : *clients*) {  
 try {  
 for (String message: *messages*) {  
 clientHandler.sendMessage(message);  
 }  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
 }  
}

Вывод программы:





**Вывод**: в результате выполнения практической работы была освоена работа с сокетами.