**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»**

**ім. Ігоря Сікорського»**

Кафедра системного проектування

ЛАБОРАТОРНА РОБОТА №5

з дисципліни

" **ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ**"

на тему: «Дослідження мережевих можливостей розподілених систем та реалізація клієнт-серверного додатку.»

Виконав:

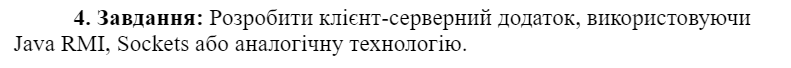
Студент ІІІ курсу

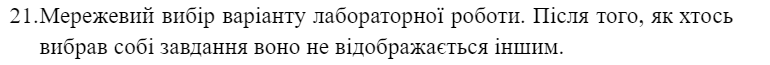
групи ДА-81

Нікітін Б. Д.

Київ-2021

# Завдання





# Опис виконання

За допомогою Selector я реалізував server. В безкінечному циклі я передивляюся всі ініціалізовані активності (Accept, Read). Та обробляю кожну з них.

ClientTCP реалізує клієнта для контакту з сервером.

У LabVariant я реалізовую контроль доступу до списку лабораторних, та функції вибирання лабораторної студентом.

Список варіантів я задаю на сервері.

Взаємодія з сервером побудована по схемі get запросів.

getAvailableVariants, getVariant, getChosenVariants.

# Результат роботи

Server is running on port : 8888

Server got connection from : /127.0.0.1:55568

Client [0] connected to localhost/127.0.0.1:8888

Client [10] connected to localhost/127.0.0.1:8888

Server got connection from : /127.0.0.1:55569

Client [3] connected to localhost/127.0.0.1:8888

Server got connection from : /127.0.0.1:55570

Client [0] sent request to localhost/127.0.0.1:8888

Request :

{

"headers": {"endpoint": "availableVariants"},

"body": {"student": "Student(0)"}

}

Server got request on endpoint :

with data :

{

"headers": {"endpoint": "availableVariants"},

"body": {"student": "Student(0)"}

}

Server processed request from : /127.0.0.1:55568

Client [0] got response from localhost/127.0.0.1:8888

Response :

{

"headers": {"status": 200},

"body": {

"variants": [

"first",

"second",

"third",

"fourth",

"fifth"

],

"info": "You have got all available variants"

}

}

Client [0] sent request to localhost/127.0.0.1:8888

Request :

{

"headers": {"endpoint": "getVariant"},

"body": {

"student": "Student(0)",

"variant": "third"

}

}

Server got request on endpoint :

with data :

{

"headers": {"endpoint": "getVariant"},

"body": {

"student": "Student(0)",

"variant": "third"

}

}

Server processed request from : /127.0.0.1:55568

Client [0] got response from localhost/127.0.0.1:8888

Response :

{

"headers": {"status": 200},

"body": {"info": "You have got :third"}

}

Client [10] sent request to localhost/127.0.0.1:8888

Request :

{

"headers": {"endpoint": "getVariant"},

"body": {

"student": "Student(10)",

"variant": "third"

}

}

Server got request on endpoint :

with data :

{

"headers": {"endpoint": "getVariant"},

"body": {

"student": "Student(10)",

"variant": "third"

}

}

Server processed request from : /127.0.0.1:55569

Client [10] got response from localhost/127.0.0.1:8888

Response :

{

"headers": {"status": 200},

"body": {"info": "This variant is already chosen"}

}

Client [0] sent request to localhost/127.0.0.1:8888

Request :

{

"headers": {"endpoint": "getVariant"},

"body": {

"student": "Student(0)",

"variant": "second"

}

}

Server got request on endpoint :

with data :

{

"headers": {"endpoint": "getVariant"},

"body": {

"student": "Student(0)",

"variant": "second"

}

}

Server processed request from : /127.0.0.1:55568

Client [0] got response from localhost/127.0.0.1:8888

Response :

{

"headers": {"status": 200},

"body": {"info": "You can have no more variant"}

}

Client [3] sent request to localhost/127.0.0.1:8888

Request :

{

"headers": {"endpoint": "getVariant"},

"body": {

"student": "Student(3)",

"variant": "third"

}

}

Server got request on endpoint :

with data :

{

"headers": {"endpoint": "getVariant"},

"body": {

"student": "Student(3)",

"variant": "third"

}

}

Server processed request from : /127.0.0.1:55570

Client [3] got response from localhost/127.0.0.1:8888

Response :

{

"headers": {"status": 200},

"body": {"info": "This variant is already chosen"}

}

Server closed socket from : /127.0.0.1:55568

Client [0] closed connection to localhost/127.0.0.1:8888

Client [10] closed connection to localhost/127.0.0.1:8888

Client [3] closed connection to localhost/127.0.0.1:8888

Server closed socket from : /127.0.0.1:55569

Server closed socket from : /127.0.0.1:55570

Результат вірний.

# Програмна реалізація

Server.java

import org.json.JSONObject;  
  
import java.io.\*;  
import java.net.\*;  
import java.nio.\*;  
import java.nio.channels.\*;  
import java.nio.charset.StandardCharsets;  
import java.util.\*;  
  
public class Server implements Runnable  
{  
 // Этот порт будет прослушиваться сервером  
 private final int port;  
  
 // Выделение памяти под буфер  
 private final ByteBuffer buffer = ByteBuffer.*allocate*( 16384 );  
  
 // Инициализация вариантов лабораторных  
 private final LabVariants<String> variants;  
  
 public Server( int port) {  
 this.port = port;  
 this.variants = new LabVariants<String>(Arrays.*asList*("first", "second", "third", "fourth", "fifth"), 1);  
 }  
  
 public void listen(){  
 new Thread( this ).start();  
 }  
  
 public void run() {  
 try {  
 // Я создаю ServerSocketChannel  
 ServerSocketChannel ssc = ServerSocketChannel.*open*();  
  
 // канал должен быть non-blocking  
 ssc.configureBlocking( false );  
  
 // Подключаем ServerSocket к каналу  
 ServerSocket serverSocket = ssc.socket();  
 InetSocketAddress isa = new InetSocketAddress( port );  
 serverSocket.bind(isa);  
  
 // Создаем новый обьект Селектор  
 Selector selector = Selector.*open*();  
  
 // Регистрируем селектор для ServerSocketChannel  
 ssc.register( selector, SelectionKey.*OP\_ACCEPT* );  
 System.*out*.println( "Server is running on port : " + port );  
  
 while (true) {  
 // Если есть активности  
 int num = selector.select();  
  
 // Если нету активностей продолжаем  
 if (num == 0) {  
 continue;  
 }  
  
 // Нужно получить ключи активностей  
 Set<SelectionKey> keys = selector.selectedKeys();  
 for (SelectionKey key : keys) {  
 // Проверка типа активности  
 if ((key.readyOps() & SelectionKey.*OP\_ACCEPT*) ==  
 SelectionKey.*OP\_ACCEPT*) {  
  
 // Прослушиваем InputStream  
 Socket socket = serverSocket.accept();  
 System.*out*.println("Server got connection from : " + socket.getRemoteSocketAddress());  
  
 // Проверка на non-blocking  
 SocketChannel sc = socket.getChannel();  
 sc.configureBlocking(false);  
  
 // регистрируем селектор на чтение  
 sc.register(selector, SelectionKey.*OP\_READ*);  
 } else if ((key.readyOps() & SelectionKey.*OP\_READ*) ==  
 SelectionKey.*OP\_READ*) {  
  
 SocketChannel sc = null;  
  
 try {  
 // Обрабатываем входные данные  
 sc = (SocketChannel) key.channel();  
 boolean status = processInput(sc);  
  
 // Если соединение закрыто, то закрываем сокет  
 if (!status) {  
 key.cancel();  
  
 Socket s = null;  
 try {  
 s = sc.socket();  
 System.*out*.println("Server closed socket from : " + s.getRemoteSocketAddress() );  
 s.close();  
 } catch (IOException ie) {  
 System.*err*.println("Server Error closing socket " + s + " : " + ie);  
 }  
 }  
 } catch (IOException ie) {  
  
 key.cancel();  
  
 try {  
 System.*out*.println("Server closed socket channel from : " + sc.getRemoteAddress());  
 sc.close();  
 } catch (IOException ie2) {  
 System.*err*.println("Server Error I/O : " + ie2);  
 }  
 }  
 }  
 }  
  
 // Удаляем ключи  
 keys.clear();  
 }  
 } catch( IOException ie ) {  
 System.*err*.println("Server Error I/O : " + ie);  
 ie.printStackTrace();  
 }  
 }  
  
 // Обработка входных данных  
 private boolean processInput( SocketChannel sc ) throws IOException {  
 // Записываем данные в буфер (которые пришли из InputStream)  
 buffer.clear();  
 sc.read( buffer );  
 buffer.flip();  
  
 // Если дата пустая то false  
 if (buffer.limit() == 0){  
 return false;  
 }  
  
 // Считываем данные записаные в буфер  
 byte[] data = buffer.array();  
  
 JSONObject requestJson = new JSONObject(new String(data, StandardCharsets.*UTF\_8*));  
 JSONObject requestHeaders = (JSONObject) requestJson.get("headers");  
 JSONObject requestBody = (JSONObject) requestJson.get("body");  
 String endpoint = (String) requestHeaders.get("endpoint");  
  
 System.*out*.println("Server got request on endpoint : " + "\nwith data :\n" + requestJson.toString(2));  
  
 JSONObject responseJson = new JSONObject();  
 JSONObject responseBody = new JSONObject();  
 JSONObject responseHeaders = new JSONObject();  
  
 // Обрабатываем варианты доступных запросов  
 switch (endpoint) {  
 case "availableVariants" -> {  
 responseHeaders.put("status", 200);  
 responseBody.put("variants", variants.getAvailableVariants());  
 responseBody.put("info", "You have got all available variants");  
 }  
 case "getVariant" -> {  
 responseHeaders.put("status", 200);  
 String student = (String) requestBody.get("student");  
 String variant = (String) requestBody.get("variant");  
 int var = variants.getVariant(student, variant);  
 switch (var){  
 case 0 -> {  
 responseBody.put("info", "You have got :" + variant);  
 }  
 case 1 -> {  
 responseBody.put("info", "There is no such variant");  
 }  
 case 2 -> {  
 responseBody.put("info", "There is no more available variants");  
 }  
 case 3 -> {  
 responseBody.put("info", "This variant is already chosen");  
 }  
 case 4 -> {  
 responseBody.put("info", "You can have no more variant");  
 }  
 }  
 }  
 }  
  
 responseJson.put("headers", responseHeaders);  
 responseJson.put("body", responseBody);  
  
 // Записываем данные в буфер и после отправляем буфер через outputStream  
 writeBuffer(sc, responseJson.toString().getBytes(StandardCharsets.*UTF\_8*));  
  
 System.*out*.println( "Server processed request from : " + sc.getRemoteAddress());  
  
 return true;  
 }  
  
 private void writeBuffer(SocketChannel sc, byte[] data) throws IOException{  
 buffer.clear();  
 buffer.put(data);  
 buffer.flip();  
  
 sc.write(buffer);  
 }  
  
 static public void main(String[] args) throws Exception {  
 int port = Integer.*parseInt*( args[0] );  
 // Запускаем сервер на порту port  
 Server server = new Server( port );  
 server.listen();  
 }  
}

ClientTCP.java

import org.json.JSONObject;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.OutputStream;  
import java.net.Socket;  
import java.nio.charset.StandardCharsets;  
import java.util.concurrent.CompletableFuture;  
import java.util.concurrent.ExecutionException;  
  
public class ClientTCP {  
  
 Socket socket;  
 InputStream inputStream;  
 OutputStream outputStream;  
  
 int clientId;  
 String serverName;  
 int port;  
  
 ClientTCP(int clientId, String serverName, int port) {  
 this.clientId = clientId;  
 this.serverName = serverName;  
 this.port = port;  
 }  
  
 public void connect() throws IOException {  
 // Создаем сокет для подальшего использования  
 socket = new Socket(serverName, port);  
 inputStream = socket.getInputStream();  
 outputStream = socket.getOutputStream();  
 System.*out*.println("Client [" + clientId + "] connected to " + socket.getRemoteSocketAddress());  
 }  
  
 public void closeConnection() throws IOException {  
 // Закрываем сокет  
 socket.close();  
 System.*out*.println("Client [" + clientId + "] closed connection to " + socket.getRemoteSocketAddress());  
 }  
  
 public JSONObject request(JSONObject Data) throws IOException, InterruptedException, ExecutionException {  
 CompletableFuture<JSONObject> result =  
 output.call(Data)  
 // Отрпавка данных через outputStream  
 .thenApply(  
 out -> {  
 if (out) {  
 try {  
 // Получаем данные из inputStream  
 return input.call().get();  
 } catch (InterruptedException | ExecutionException e) {  
 e.printStackTrace();  
 }  
 }  
 return null;  
 });  
 return result.get();  
 }  
  
 @FunctionalInterface  
 interface jsonInput {  
 CompletableFuture<JSONObject> call();  
 }  
  
 @FunctionalInterface  
 interface jsonOutput {  
 CompletableFuture<Boolean> call(JSONObject Data);  
 }  
  
 jsonInput input = () -> CompletableFuture  
 .*supplyAsync*(  
 () -> {  
 JSONObject result = null;  
 float count = -1;  
 while (count == -1) {  
 try {  
 byte[] data = new byte[1000];  
 // Прием данных из InputStream  
 count = inputStream.read(data);  
 if (count == -1) {  
 // Если данные пустые мы пробуем еще раз  
 continue;  
 }  
 // Записываем данные в JSONObject  
 result = new JSONObject(new String(data, StandardCharsets.*UTF\_8*));  
 System.*out*.println("Client [" + clientId + "] got response from " + socket.getRemoteSocketAddress() + "\n" + "Response :\n" + result.toString(2));  
 } catch (IOException e) {  
 System.*err*.println("Client [" + clientId + "] Error : " + e);  
 result = null;  
 }  
 }  
 // Возвращаем response  
 return result;  
 }  
 );  
  
 jsonOutput output = (Data) -> CompletableFuture  
 .*supplyAsync*(  
 () -> {  
 try {  
 // Считываем данные из inputStream  
 System.*out*.println("Client [" + clientId + "] sent request to " + socket.getRemoteSocketAddress() + "\n" + "Request :\n" + Data.toString(2));  
 outputStream.write(Data.toString().getBytes(StandardCharsets.*UTF\_8*));  
 return true;  
 } catch (IOException e) {  
 System.*err*.println("Client [" + clientId + "] Error : " + e);  
 return false;  
 }  
 }  
 );  
}

LabVariants.java

import java.util.\*;  
  
public class LabVariants<T>{  
 Map<String, List<T>> variants = new HashMap<>();  
 int max;  
  
  
 LabVariants(List<T> variants, int max){  
 this.variants.put("available", variants);  
 this.variants.put("all", variants);  
 this.max = max;  
 }  
  
 int getVariant(String student, T variant){  
 if (variants.get("all").contains(variant)){  
 // Если вариант существует  
 if (variants.get("available").size() == 0){  
 // Если нету доступных вариантов  
 return 2;  
 }  
 if (variants.containsKey("chosen")) {  
 if (variants.get("chosen").contains(variant)) {  
 // Если вариант уже выбран  
 return 3;  
 }  
 if (variants.containsKey(student)){  
 // Если у студента уже есть варианты  
 if (variants.get(student).size() == max){  
 // Если количество вариантов студента достигло максимума  
 return 4;  
 }  
 // Если все ок  
 variants.get(student).add(variant);  
 variants.get("chosen").add(variant);  
 variants.get("available").remove(variant);  
 return 0;  
 }  
 // Если все ок  
 ArrayList<T> studentVariants = new ArrayList<>();  
 studentVariants.add(variant);  
 variants.put(student, studentVariants);  
 variants.get("chosen").add(variant);  
 variants.get("available").remove(variant);  
 return 0;  
 }  
 // Если все ок  
 List<T> studentVariants = new ArrayList<>();  
 studentVariants.add(variant);  
 variants.put(student, studentVariants);  
 variants.put("chosen", studentVariants);  
 ArrayList<T> s = new ArrayList<>(variants.get("available"));  
 for (int i = 0; i < s.size(); i++){  
 T v = s.get(i);  
 // Удаляем вариант из available  
 if (v.equals(variant)){  
 s.remove(i);  
 break;  
 }  
 }  
 variants.put("available", s);  
 return 0;  
 }  
 return 1;  
 }  
  
 List<T> getAvailableVariants(){  
 return variants.get("available");  
 }  
  
 List<T> getChosenVariants(){  
 return variants.get("chosen");  
 }  
  
 T getStudentVariant(String student){  
 return variants.containsKey(student) ? (T) variants.get(student) : null;  
 }  
  
  
}

Main.java

import org.json.JSONArray;  
import org.json.JSONObject;  
  
import java.io.IOException;  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.concurrent.ExecutionException;  
  
public class Main {  
 static public void main(String[] args) throws Exception {  
 Map<ClientTCP, String> studentsVariants = new HashMap<>();  
 String serverName = args[0];  
 int port = Integer.*parseInt*(args[1]);  
  
 Server server = new Server(port);  
 server.listen();  
  
 Thread.*sleep*(2000);  
  
 ClientTCP[] clientList = new ClientTCP[]  
 {  
 new ClientTCP(0, serverName, port),  
 new ClientTCP(10, serverName, port),  
 new ClientTCP(3, serverName, port),  
  
 };  
 for (ClientTCP client : clientList){  
 try {  
 // Подключаем наших клиентов к серверу  
 client.connect();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 try {  
 // Делаем запрос, чтоб получить все варианты  
 JSONObject availableRequest = clientList[0].request(*createRequest*("{endpoint : availableVariants}", "{student : " + "Student(" + clientList[0].clientId + ")}"));  
 JSONObject requestBody = *parse*(availableRequest, "body");  
 JSONArray variantsList = requestBody.getJSONArray("variants");  
 int variantIndex = (int) (Math.*random*() \* (variantsList.length()));  
 int variantIndex2 = (int) (Math.*random*() \* (variantsList.length()));  
 // Делаем запрос, чтоб client[0] получил вариант  
 clientList[0].request(*createRequest*("{endpoint : getVariant}", "{variant : " + variantsList.get(variantIndex) + "," + "student : " + "Student(" + clientList[0].clientId + ")}"));  
 // Делаем запрос, чтоб client[10] получил вариант  
 clientList[1].request(*createRequest*("{endpoint : getVariant}", "{variant : " + variantsList.get(variantIndex) + "," + "student : " + "Student(" + clientList[1].clientId + ")}"));  
 // Делаем запрос, чтоб client[0] получил вариант  
 clientList[0].request(*createRequest*("{endpoint : getVariant}", "{variant : " + variantsList.get(variantIndex2) + "," + "student : " + "Student(" + clientList[0].clientId + ")}"));  
 // Делаем запрос, чтоб client[3] получил вариант  
 clientList[2].request(*createRequest*("{endpoint : getVariant}", "{variant : " + variantsList.get(variantIndex) + "," + "student : " + "Student(" + clientList[2].clientId + ")}"));  
 } catch (IOException | InterruptedException | ExecutionException e) {  
 e.printStackTrace();  
 }  
 for (ClientTCP client : clientList){  
 try {  
 // Отключаем клиентов от сервера  
 client.closeConnection();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
  
 static JSONObject createRequest(String headers, String data){  
 JSONObject requestHeaders = new JSONObject(headers);  
 JSONObject requestBody = new JSONObject(data);  
 JSONObject request = new JSONObject();  
 request.put("headers", requestHeaders);  
 request.put("body", requestBody);  
 return request;  
 }  
  
 static JSONObject parse(JSONObject request, String tag){  
 return (JSONObject) request.get(tag);  
 }  
}