设计报告

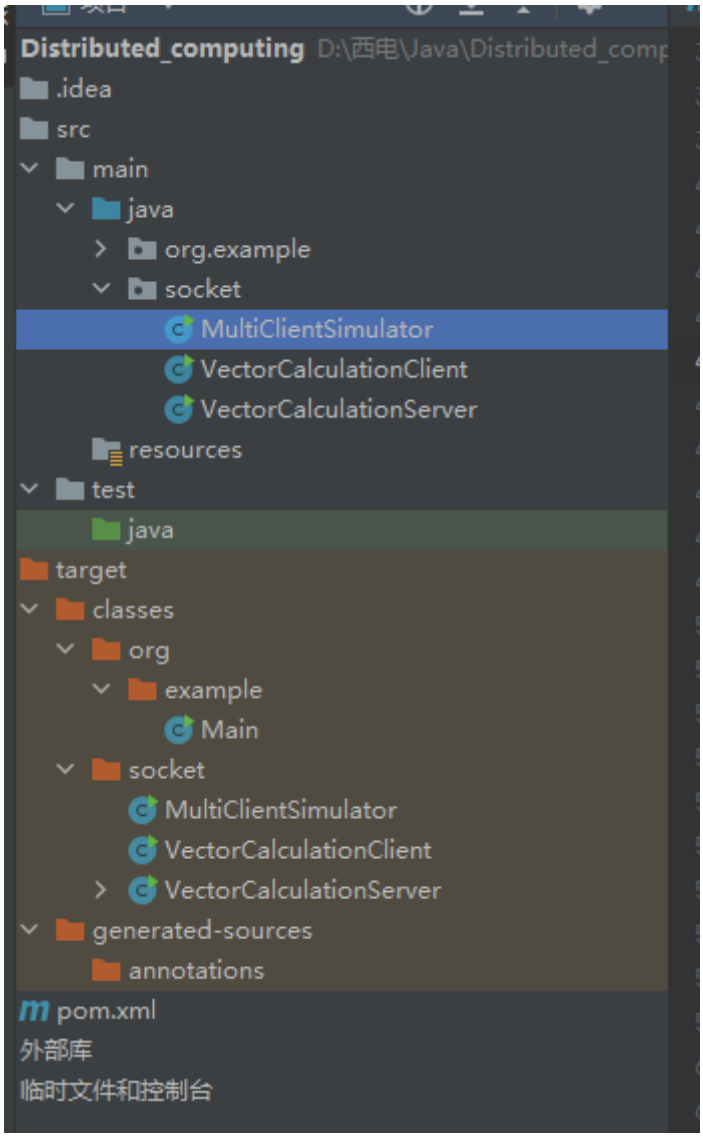
引言

本项目旨在开发一个基于Socket的向量计算服务器程序，支持多客户端并发访问。服务器能够处理来自客户端的向量点乘和叉乘计算请求。

系统架构

系统包括两个主要组件：服务器端和客户端。

- **服务器端** 监听特定端口上的连接请求，接受来自客户端的向量运算请求，并执行相应的计算，然后将结果返回给客户端。
- **客户端** 向服务器发送向量计算请求，并接收计算结果。



详细设计

- **服务器端设计**:
  - 使用`ServerSocket`在指定端口上监听传入的连接请求。
  - 为每个接受的连接创建一个新的`ClientHandler`线程，以实现多客户端并发处理。
  - `ClientHandler`线程读取客户端发送的数据，根据请求类型（点乘或叉乘）执行相应的计算，并将结果发送回客户端。
- **客户端设计**:
  - 通过`Socket`连接到服务器端，并发送请求（包括操作类型和向量数据）。
  - 接收并显示服务器返回的计算结果。

服务器端&客户端设计代码

- **服务器端代码**

```
package socket;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Arrays;

class VectorCalculationServer {
    public static void main(String[] args) {
        int portNumber = 8888; // 服务器监听的端口号

        try {
            ServerSocket serverSocket = new ServerSocket(portNumber);
            System.out.println("服务器启动，监听端口：" + portNumber);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("接收到新的客户端连接");
                ClientHandler clientHandler = new
ClientHandler(clientSocket);
                clientHandler.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static class ClientHandler extends Thread {
        private Socket clientSocket;

        public ClientHandler(Socket socket) {
            this.clientSocket = socket;
        }

        public void run() {
            try (
                    PrintWriter out = new
PrintWriter(clientSocket.getOutputStream(), true);
                    BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()))
            ) {
```

```
                String inputLine;
                while ((inputLine = in.readLine()) != null) {
                    String[] input = inputLine.split(" ");
                    if (input.length != 3) {
                        out.println("Invalid input");
                        continue;
                    }

                    String operation = input[0];
                    String[] vectorA = input[1].split(",");
                    String[] vectorB = input[2].split(",");

                    if (operation.equals("dot")) {
                        if (vectorA.length != vectorB.length) {
                            out.println("Invalid input lengths");
                            continue;
                        }
                        double[] vA =
Arrays.stream(vectorA).mapToDouble(Double::parseDouble).toArray();
                        double[] vB =
Arrays.stream(vectorB).mapToDouble(Double::parseDouble).toArray();
                        double dotProduct = dotProduct(vA, vB);
                        out.println("Dot product: " + dotProduct);
                    } else if (operation.equals("cross")) {
                        if (vectorA.length != 3 || vectorB.length != 3) {
                            out.println("Cross product requires 3-dimensional
vectors");
                            continue;
                        }
                        double[] vA =
Arrays.stream(vectorA).mapToDouble(Double::parseDouble).toArray();
                        double[] vB =
Arrays.stream(vectorB).mapToDouble(Double::parseDouble).toArray();
                        double[] crossProduct = crossProduct(vA, vB);
                        out.println("Cross product: " +
Arrays.toString(crossProduct));
                    } else {
                        out.println("Invalid operation");
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            } finally {
                try {
```

```
                    clientSocket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        private double dotProduct(double[] a, double[] b) {
            double result = 0;
            for (int i = 0; i < a.length; i++) {
                result += a[i] * b[i];
            }
            return result;
        }

        private double[] crossProduct(double[] a, double[] b) {
            return new double[]{
                    a[1] * b[2] - a[2] * b[1],
                    a[2] * b[0] - a[0] * b[2],
                    a[0] * b[1] - a[1] * b[0]
            };
        }
    }
}
```

- **客户端代码**

```
package socket;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class VectorCalculationClient {
    public static void main(String[] args) {
        String hostName = "localhost"; // 服务器主机名
        int portNumber = 8888; // 服务器端口号

        try (
                Socket socket = new Socket(hostName, portNumber);
```

```
                PrintWriter out = new PrintWriter(socket.getOutputStream(),
true);
                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                BufferedReader stdIn = new BufferedReader(new
InputStreamReader(System.in))
        ) {
            String userInput;
            while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                String response = in.readLine();
                System.out.println("服务器响应: " + response);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

技术栈

- Java语言
- Java Socket编程

安全和异常处理

- 异常处理机制确保服务器和客户端在网络断开或数据传输错误时能够正确响应。
- 对输入数据进行验证，确保格式正确且适合计算。