Por Que Não Paralelizar?

João Pedro da Silva Zanirati Nunes – 202209081405
Polo Azenha
Nível 5: Por Que Não Paralelizar? – 9003 – Mundo 3

Objetivo da Prática

O objetivo é criar um sistema de cadastro, onde o servidor gerencia as informações de produtos e usuários. Os clientes podem interagem com o servidor para visualizar os produtos, fazer login e executar operações de entrada/saída de produtos. Usando threads e uma interface gráfica busca melhorar a eficiência e a experiência de um possivel usuário.

1º Procedimento | Criando o Servidor e Cliente de Teste

```
package cadastroserver;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class CadastroServer {

    public static void main(String[] args) throws IOException {

        ServerSocket serverSocket = new ServerSocket(4321);

        System.out.println("Servidor iniciado. Aguardando conexões...");

        while (true) {

            Socket clientSocket = serverSocket.accept();

            System.out.println("Cliente conectado: " +
clientSocket.getInetAddress().getHostAddress());
```

```
                                BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
            String login = in.readLine();
            String senha = in.readLine();
            if (validarCredenciais(login, senha)) {
                out.println("Credenciais válidas. Aguardando requisições...");
                while (true) {
                    String requisicao = in.readLine();
                    if (requisicao.equals("L")) {
                        String produtos = obterConjuntoProdutos();
                        out.println(produtos);
                    } else {
                        out.println("Requisição inválida");
                    }
                }
            } else {
                out.println("Credenciais inválidas. Desconectando...");
                clientSocket.close();
            }
        }
    }
}
```

```
package cadastroserver;

import controller.ProdutoJpaController;

import controller.UsuarioJpaController;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.Socket;

public class CadastroThread extends Thread {

    private ProdutoJpaController ctrl;

    private UsuarioJpaController ctrlUsu;

    private Socket s1;

    private ObjectOutputStream out;

    private ObjectInputStream in;

        public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu,
Socket s1) {

        this.ctrl = ctrl;

        this.ctrlUsu = ctrlUsu;

        this.s1 = s1;

        try {

            this.out = new ObjectOutputStream(s1.getOutputStream());

            this.in = new ObjectInputStream(s1.getInputStream());

        } catch (IOException e) {

           System.out.println(e);

        }

    }
```

```
@Override
public void run() {
    try {
        String login = (String) in.readObject();
        String senha = (String) in.readObject();
        Integer id =(Integer) in.readObject();
        if (validarCredenciais(login, senha, id)) {
            out.writeObject("Bem-vindo ao servidor!");
            while (true) {
                String comando = (String) in.readObject();
                if ("L".equals(comando)) {
                    enviarConjuntoProdutos();
                } else if ("E".equals(comando) || "S".equals(comando)) {
                    processarMovimento(comando);
                } else if ("Q".equals(comando)) {
                    break;
                }
            }
        } else {
            out.writeObject("Credenciais inválidas. Desconectando...");
            s1.close();
        }
    } catch (IOException | ClassNotFoundException e) {
      System.out.println(e);
```

```
        }
    }
    private boolean validarCredenciais(String login, String senha, Integer id) {
        return ctrlUsu.findUsuario(login, senha, id) != null;
    }
    private void enviarConjuntoProdutos() {
        try {
            out.writeObject(ctrl.listaDeProdutos());
        } catch (IOException e) {
            System.out.println(e);
        }
    }
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
public class Main {
    public static void main(String[] args) {
                                                    EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");
```

```
        ProdutoJpaController ctrl = new ProdutoJpaController(emf);
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
        ServerSocket serverSocket = null;
        try {
            int port = 4321;
            serverSocket = new ServerSocket(port);
            System.out.println("Servidor aguardando conexões na porta " + port);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                        CadastroThread cadastroThread = new CadastroThread(ctrl, ctrlUsu,
clientSocket);
                cadastroThread.start();
            }
        } catch (IOException e) {
            System.out.println(e);
        } finally {
            if (emf != null && emf.isOpen()) {
                emf.close();
            }
        }
    }
}
```

Como funcionam as classes Socket e ServerSocket?

Socket: Representa um ponto de conexão para comunicação entre os programas, sendo usado para enviar e receber dados entre cliente e servidor.

ServerSocket: Utilizado no lado do servidor para aguardar e aceitar conexões de clientes, criando instâncias de Socket para cada conexão aceita.

Qual a importância das portas para a conexão com servidores?

Permitindo a identificação e comunicação entre os aplicativos em um dispositivo com a rede. São essenciais para a comunicação eficiente, as portas facilitam a coexistência de várias aplicações em um mesmo dispositivo, padronizam a comunicação, contribuem para a segurança e direcionam o tráfego na rede.

Para que servem as classes de entrada e saída ObjectInputStream e

ObjectOutputStream, e por que os objetos transmitidos devem ser

serializáveis?

São usados para converter objetos Java em sequências de bytes e reconstruí-los posteriormente. Os objetos transmitidos devem implementar a interface Serializable para serem serializáveis. Isso é essencial para comunicação entre processos distribuídos em Java.

Por que, mesmo utilizando as classes de entidades JPA no cliente, foi

possível garantir o isolamento do acesso ao banco de dados?

O isolamento ocorre porque o cliente interage com o servidor, que gerencia e valida as operações de banco de dados. Isso garante segurança e controle, impedindo que o cliente acesse diretamente o banco de dados e execute operações não autorizadas. A arquitetura cliente-servidor, aliada ao uso de JPA no servidor, proporciona uma camada de abstração para facilitar a segurança e manutenção do acesso ao banco de dados.

2º Procedimento | Servidor Completo e Cliente Assíncrono

```
package cadastroclient;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.net.Socket;

import java.util.List;


public class CadastroClient {

   public static void main(String[] args) {

      try {

         Socket socket = new Socket("localhost", 4321);

         ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());

         ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

         String login = "op1";

         String senha = "op1";

         out.writeObject(login);

         out.writeObject(senha);

         out.writeObject("L");

         List<String> listaDeEntidades = (List<String>) in.readObject();

         System.out.println("Entidades recebidas:");

         for (String entidade : listaDeEntidades) {

            System.out.println(entidade);

         }
```

```
            socket.close();
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
        }
    }
}
package cadastroclient;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JTextArea;

public class CadastroClientV2 {
public static void main(String[] args) {
    try (Socket socket = new Socket("localhost", 4321);
         ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
         ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {
        out.writeObject("op1");
        out.writeObject("op1");
        out.writeObject(1);
```

```
        SaidaFrame saidaFrame = new SaidaFrame();
        JTextArea texto = saidaFrame.texto;
        ThreadClient preenchimentoAsync = new ThreadClient(in, texto);
        preenchimentoAsync.start();

        OUTER:
        while (true) {
            mostrarMenu();
            String comando = lerComandoTeclado();
            if (null == comando) {
                saidaFrame.exibirMensagem("Comando inválido. Tente novamente.");
            } else {
                switch (comando) {
                    case "X" -> {
                        break OUTER;
                    }
                    case "L" -> out.writeObject("L");
                    case "E", "S" -> enviarComandoES(comando, out);
                            default -> saidaFrame.exibirMensagem("Comando inválido. Tente
novamente.");
                }
            }
        }
        preenchimentoAsync.join();
    } catch (IOException e) {
```

```
            System.out.println(e);
        } catch (InterruptedException ex) {
            Logger.getLogger(CadastroClientV2.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    private static void mostrarMenu() {
        System.out.println("Menu:");
        System.out.println("L - Listar");
        System.out.println("E - Entrada");
        System.out.println("S - Saída");
        System.out.println("X - Finalizar");
    }
    private static String lerComandoTeclado() throws IOException {
                BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Escolha um comando: ");
        return reader.readLine().toUpperCase();
    }
    private static void enviarComandoES(String comando, ObjectOutputStream out) throws IOException {
                try (BufferedReader reader = new BufferedReader(new InputStreamReader(System.in))) {
            while (true) {
                System.out.print("Id da pessoa: ");
                out.writeObject(Integer.valueOf(reader.readLine()));
```

```
            System.out.print("Id do produto: ");
            out.writeObject(Integer.valueOf(reader.readLine()));

            System.out.print("Quantidade: ");
            out.writeObject(Integer.valueOf(reader.readLine()));

            System.out.print("Valor unitário: ");
            out.writeObject(Double.valueOf(reader.readLine()));

            System.out.print("Deseja adicionar mais um item? (S/N): ");
            String continuar = reader.readLine().toUpperCase();
            if (!"S".equals(continuar)) {
               break;
            }
         }
      }
   }
}

package cadastroclient;
import java.io.IOException;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;
import java.io.ObjectInputStream;
```

```
public class ThreadClient extends Thread {
    private final ObjectInputStream entrada;
    private final JTextArea textArea;
    public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {
        this.entrada = entrada;
        this.textArea = textArea;
    }
    @Override
    public void run() {
        try {
            while (true) {
                Object objetoRecebido = entrada.readObject();
                SwingUtilities.invokeLater(() -> {
                    if (objetoRecebido instanceof String string) {
                        textArea.append(string + "\n");
                    } else if (objetoRecebido instanceof java.util.List) {
                                    java.util.List<String> listaProdutos = (java.util.List<String>)
objetoRecebido;
                        for (String produto : listaProdutos) {
                            textArea.append(produto + "\n");
                        }
                    }
                });
            }
```

```
        } catch (IOException | ClassNotFoundException e) {
            System.out.println(e);
        }
    }
}
package cadastroclient;
import javax.swing.JDialog;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;

public class SaidaFrame extends JDialog {
    public JTextArea texto;
    private SaidaFrame saidaFrame;
    public SaidaFrame() {
        setBounds(100, 100, 400, 300);
        setModal(false);
        texto = new JTextArea();
        texto.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(texto);
        getContentPane().add(scrollPane);
        setTitle("Saída do Cliente");
        setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    }
    public class MensagemJanela {
        public MensagemJanela() {
```

```
            saidaFrame = new SaidaFrame();
        }
    }
    public void exibirMensagem(String mensagem) {
        saidaFrame.texto.append(mensagem + "\n");
        saidaFrame.setVisible(true);
    }
}
```

Como as Threads podem ser utilizadas para o tratamento assíncrono das

respostas enviadas pelo servidor?

As Threads permitem tratar respostas do servidor de forma assíncrona, garantindo que o programa continue executando outras tarefas enquanto aguarda as respostas.

Para que serve o método invokeLater, da classe SwingUtilities?

Serve para agendar a execução de tarefas na Event Dispatch Thread (EDT) do Swing, garantindo a atualização segura e consistente da interface gráfica.

Como os objetos são enviados e recebidos pelo Socket Java?

No lado do cliente, os objetos são escritos em um ObjectOutputStream associado ao OutputStream do Socket. No lado do servidor, os objetos são lidos de um ObjectInputStream associado ao InputStream do Socket.

Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento.

Comportamento Síncrono:

- Bloqueia a execução da thread até a conclusão da operação.
- Aguarda a resposta do servidor antes de continuar.
- Mais simples de implementar.

Comportamento Assíncrono:

- Não bloqueia a thread principal, permitindo a execução de outras tarefas.
- Utiliza callbacks, promessas ou mecanismos semelhantes.
- Pode resultar em melhor desempenho e maior responsividade.

Conclusão

Ao desenvolver a aplicação cliente-servidor em Java com sockets e JPA, pude explorar de forma prática conceitos fundamentais de comunicação distribuída. A utilização de threads e objetos serializáveis contribuiu para uma implementação mais eficiente e interativa. A experiência ressalta a importância do tratamento assíncrono, proporcionando uma compreensão mais profunda sobre a fluidez na interação do usuário. Essa prática consolidou meu entendimento sobre o funcionamento dessas tecnologias e sua aplicação em cenários do mundo real.