Por que não paralelizar?

Jonathan da Silva Araujo – 202205178111

Polo: Nova Iguaçu – RJ
Por que não paralelizar – 9001 - 2023.2

Objetivo da Prática

1. Criar servidores Java com base em Sockets.
2. Criar clientes síncronos para servidores com base em Sockets.
3. Criar clientes assíncronos para servidores com base em Sockets.
4. Utilizar Threads para implementação de processos paralelos.

1º Procedimento – Criando o Servidor e Cliente de Teste

CadastroServer:

```java
package cadastroserver;

import controller.MovimentosJpaController;
import controller.PessoasJpaController;
import controller.ProdutosJpaController;
import controller.UsuariosJpaController;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class CadastroServer {

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     */
    public static void main(String[] args) throws IOException {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
        ProdutosJpaController ctrlProd = new ProdutosJpaController(emf);
        UsuariosJpaController ctrlUsu = new UsuariosJpaController(emf);
        MovimentosJpaController ctrlMov = new MovimentosJpaController(emf);
        PessoasJpaController ctrlPessoa = new PessoasJpaController(emf);

        try (ServerSocket serverSocket = new ServerSocket( port: 4321)) {
            System.out.println("Servidor aguardando conexoes na porta 4321...");

            while (true) {
                Socket socket = serverSocket.accept();
                CadastroThreadV2 thread = new CadastroThreadV2(ctrlUsu, ctrlMov, ctrlProd, ctrlPessoa, usuarioAutenticado: null, socket);
                thread.start(); // Inicia a thread
                System.out.println("thread iniciado!");
            }

        }
    }
}
```

CadastroThread:

```java
package cadastroserver;

import controller.ProdutosJpaController;
import controller.UsuariosJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Produtos;
import model.Usuarios;

1 usage
public class CadastroThread extends Thread {
    2 usages
    private final ProdutosJpaController ctrl;
    2 usages
    private final UsuariosJpaController ctrlUsu;
    6 usages
    private final Socket s1;

    no usages
    public CadastroThread(ProdutosJpaController ctrl, UsuariosJpaController ctrlUsu, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
    }

    @Override
public void run() {
    try (ObjectOutputStream out = new ObjectOutputStream(s1.getOutputStream());
        ObjectInputStream in = new ObjectInputStream(s1.getInputStream())){

        String login = (String) in.readObject();
        String senha = (String) in.readObject();
        List<Usuarios> usuariosList = ctrlUsu.findUsuariosEntities();
        Usuarios usuarioAutenticado = null;

        for (Usuarios usuario : usuariosList) {
            if (usuario.getLogin().equals(login) && usuario.getSenha().equals(senha)) {
                usuarioAutenticado = usuario;
                break;
            }
        }

        if (usuarioAutenticado == null) {
            System.out.println("Credenciais inválidas. Desconectando cliente.");
            return;
        }

        System.out.println("Usuario autenticado: " + usuarioAutenticado.getLogin());

        while (true) {
            String comando =(String) in.readObject();

            if (comando.equals("L")) {
                List<Produtos> produtos = ctrl.findProdutosEntities();
                out.writeObject(produtos);
                System.out.println("Enviando lista de produtos para o cliente.");
                break;
            }
```

```
59          }
60
61      try {
62          if (out != null) {
63              out.close();
64          }
65          if (in != null) {
66              in.close();
67          }
68          if (s1 != null && !s1.isClosed()) {
69              s1.close();
70          }
71      } catch (IOException ex) {
72          System.err.println("Erro ao fechar os fluxos e o socket: " + ex.getMessage());
73      }
74
75      } catch (IOException ex) {
76          System.err.println("Erro de comunicação: " + ex.getMessage());
77      }   catch (ClassNotFoundException ex) {
78              Logger.getLogger(CadastroThread.class.getName()).log(Level.SEVERE,  msg: null, ex);
79          }
80      }
81  }
82
```

CadastroClient:

```java
package cadastroclient;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import model.Produtos;

public class CadastroClient {

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     * @throws java.lang.ClassNotFoundException
     */
    public static void main(String[] args)throws IOException, ClassNotFoundException {
        try (Socket socket = new Socket( host: "localhost", port: 4321);
             ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {

            out.writeObject("op1");
            out.writeObject("op1");
            out.writeObject("L");
            System.out.println("Usuario conectado com sucesso");

            List<Produtos> produtos = (List<Produtos>) in.readObject();
            for (Produtos produto : produtos) {
                System.out.println(produto.getNome());
            }
        }
    }
}
```

Resultado da execução:

```
run:
Servidor aguardando conexoes na porta 4321...
thread iniciado!
[EL Info]: 2023-09-14 04:07:24.69--ServerSession(1517433692)--EclipseL
Usuario autenticado: op1
Enviando lista de produtos para o cliente.
```

```
run:
Usuario conectado com sucesso
Banana
Laranja
Manga
Batata
BUILD SUCCESSFUL (total time: 1 second)
```

a) Como funcionam as classes Socket e ServerSocket?

A classe `ServerSocket` atua como um ouvinte que aguarda conexões de clientes em um servidor. Quando uma conexão de cliente é estabelecida, o `ServerSocket` cria um novo objeto `Socket` para lidar com a comunicação com esse cliente específico. Em resumo, essas duas classes são componentes fundamentais para a implementação de comunicação entre clientes e servidores em Java. O `ServerSocket` gerencia a espera por conexões de clientes, enquanto os objetos `Socket` são usados para a comunicação real com cada cliente conectado.

b) Qual a importância das portas para a conexão com servidores?

Portas são números de identificação atribuídos aos processos de comunicação em um servidor. Elas desempenham um papel crucial na roteamento de dados para os serviços apropriados em um servidor, permitindo que múltiplos serviços operem simultaneamente no mesmo endereço IP. Essas portas asseguram que os dados sejam entregues ao aplicativo de destino correto, facilitando, assim, a comunicação eficiente entre clientes e serviços específicos.

c) Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?

`ObjectOutputStream` tem a capacidade de transformar objetos em uma sequência de bytes, tornando-os aptos para serem transmitidos através de redes. Já o `ObjectInputStream` executa a operação oposta, revertendo essa sequência de bytes recebida de volta em objetos.

É importante destacar que os objetos que são transmitidos precisam ser serializáveis, porque a serialização garante que eles sejam convertidos em um formato de bytes padronizado. Esse formato padronizado pode, então, ser transmitido através da rede e reconstruído como objetos idênticos no lado do receptor.

d) Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?

A isolação do acesso ao banco de dados é atingida por meio da aplicação do padrão de arquitetura Cliente-Servidor e da abstração oferecida pelas classes de entidades JPA. Essas classes de entidades JPA possibilitam que o cliente interaja com os dados de maneira orientada a objetos, ao mesmo tempo em que a lógica de acesso ao banco de dados é gerenciada no servidor.

2º Procedimento – Alimentando a Base

CadastroThreadV2:

```java
package cadastroserver;

import controller.MovimentosJpaController;
import controller.PessoasJpaController;
import controller.ProdutosJpaController;
import controller.UsuariosJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.Movimentos;
import model.Produtos;
import model.Pessoas;
import model.Usuarios;
4 usages
public class CadastroThreadV2 extends Thread {
    4 usages
    private final UsuariosJpaController ctrlUsu;
    4 usages
    private final MovimentosJpaController ctrlMov;
    6 usages
    private final ProdutosJpaController ctrlProd;
    3 usages
    private final PessoasJpaController ctrlPessoa;
    4 usages
    private Usuarios usuarioAutenticado;
    3 usages
    private final Socket s1;

    1 usage
    public CadastroThreadV2(UsuariosJpaController ctrlUsu, MovimentosJpaController ctrlMov, ProdutosJpaController ctrlProd,
                            PessoasJpaController ctrlPessoa, Usuarios usuarioAutenticado, Socket s1) {
        this.ctrlUsu = ctrlUsu;
        this.ctrlMov = ctrlMov;
        this.ctrlProd = ctrlProd;
        this.ctrlPessoa = ctrlPessoa;
        this.s1 = s1;
    }

    @Override
    public void run() {
        String mensagemAutenticacao = "";
        boolean menu = true;
        try (ObjectOutputStream out = new ObjectOutputStream(s1.getOutputStream());
             ObjectInputStream in = new ObjectInputStream(s1.getInputStream())) {

            String login = (String) in.readObject();
            String senha = (String) in.readObject();

            List<Usuarios> usuariosList = ctrlUsu.findUsuariosEntities();
            for (Usuarios usuario : usuariosList) {
                if (usuario.getLogin().equals(login) && usuario.getSenha().equals(senha)) {
                    usuarioAutenticado = usuario;
                    mensagemAutenticacao = "Usuario conectado com sucesso";
                    break;
                }
            }

            if (usuarioAutenticado == null) {
                System.out.println("Credenciais inválidas. Desconectando cliente.");
                mensagemAutenticacao = "Credenciais inválidas. Desconectando cliente.";
```

```
            menu = false;
        }

        String dataHora = new SimpleDateFormat( pattern: "E MMM dd HH:mm:ss z yyyy").format(new Date()); // Obtém a data e hora formatada
        String mensagemCompleta = ">> Nova comunicação em " + dataHora  ;

        out.writeObject(mensagemCompleta);
        out.writeObject(mensagemAutenticacao);
        out.flush();

        while (menu == true) {
            String comando = (String) in.readObject();
            switch (comando) {
                case "E":
                    System.out.println("opçao E");
                    processEntrada(in, out);
                    break;
                case "S":
                    System.out.println("opçao S");
                    processSaida(in, out);
                    break;
                case "L":
                    List<Produtos> produtosList = ctrlProd.findProdutosEntities();
                    dataHora = new SimpleDateFormat( pattern: "E MMM dd HH:mm:ss z yyyy").format(new Date());
                    mensagemCompleta = ">> Nova comunicação em " + dataHora  ;
                    out.writeObject(mensagemCompleta);
                    for (Produtos produto : produtosList) {
                        String produtoInfo = produto.getNome() + " : " + produto.getQuantidade();
                        out.writeObject(produtoInfo);
                        out.flush();
                    }
                    break;
                case "X":
                    menu = false;
                    break;
                default:

                    break;
            }
        }
    } catch (IOException ex) {
        System.err.println("Erro de comunicação: " + ex.getMessage());
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(CadastroThreadV2.class.getName()).log(Level.SEVERE,  msg: null, ex);
    } catch (Exception ex) {
        Logger.getLogger(CadastroThreadV2.class.getName()).log(Level.SEVERE,  msg: null, ex);
    }
}

1 usage
private void processEntrada(ObjectInputStream in, ObjectOutputStream out) throws IOException, ClassNotFoundException, Exception {
    char tipoMovimento = 'E'; // Tipo de movimento Entrada

    int idPessoa = in.readInt();
    int idProduto = in.readInt();
    int quantidade = in.readInt();
    float valorUnitario = in.readFloat();
    int idUsuario = usuarioAutenticado.getIdUsuario();

    Usuarios usuario = ctrlUsu.findUsuarios(idUsuario);
    Pessoas pessoa = ctrlPessoa.findPessoas(idPessoa);
    Produtos produto = ctrlProd.findProdutos(idProduto);

```

```
        Movimentos movimento = new Movimentos();
        movimento.setIdMovimentos(getNextMovimentoId());
        movimento.setIdUsuario(usuario);
        movimento.setTipo(tipoMovimento);
        movimento.setIdPessoa(pessoa);
        movimento.setIdProduto(produto);
        movimento.setQuantidade(quantidade);
        movimento.setPrecoUnitario( valorUnitario);

        ctrlMov.create(movimento);

        produto.setQuantidade(produto.getQuantidade() + quantidade);
        ctrlProd.edit(produto);

        out.writeObject("Movimento de Entrada registrado com sucesso.");
    }

    1 usage
    private void processSaida(ObjectInputStream in, ObjectOutputStream out) throws IOException, ClassNotFoundException, Exception {
        char tipoMovimento = 'S'; // Tipo de movimento Saída

        int idPessoa = in.readInt();
        int idProduto = in.readInt();
        int quantidade = in.readInt();
        float valorUnitario = in.readFloat();
        int idUsuario = usuarioAutenticado.getIdUsuario();

        Usuarios usuario = ctrlUsu.findUsuarios(idUsuario);
        Pessoas pessoa = ctrlPessoa.findPessoas(idPessoa);
        Produtos produto = ctrlProd.findProdutos(idProduto);

        Movimentos movimento = new Movimentos();
        movimento.setIdMovimentos(getNextMovimentoId());
            movimento.setIdUsuario(usuario);
            movimento.setTipo(tipoMovimento);
            movimento.setIdPessoa(pessoa);
            movimento.setIdProduto(produto);
            movimento.setQuantidade(quantidade);
            movimento.setPrecoUnitario(valorUnitario);

            ctrlMov.create(movimento);

            produto.setQuantidade(produto.getQuantidade() - quantidade);
            ctrlProd.edit(produto);

            out.writeObject("Movimento de Saída registrado com sucesso.");
        }

        2 usages
        private synchronized int getNextMovimentoId() {
        List<Movimentos> movimentos = ctrlMov.findMovimentosEntities();
        int lastMovimentoId = 0;

        for (Movimentos movimento : movimentos) {
            if (movimento.getIdMovimentos() > lastMovimentoId) {
                lastMovimentoId = movimento.getIdMovimentos();
            }
        }

        return lastMovimentoId + 1;
        }

}
```

CadastroClientV2:

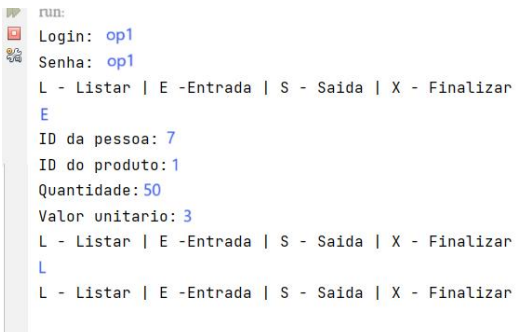
```java
package cadastroclientv2;

import ...

public class CadastroClientV2 {
    9 usages
    public CadastroClientV2() {
    }

    public static void main(String[] args) throws IOException {
        Socket socket = new Socket( host: "localhost",  port: 4321);
        ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
        ObjectInputStream in = new ObjectInputStream(socket.getInputStream());
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        boolean menu = true;
        System.out.print("Login: ");
        String login = reader.readLine();
        System.out.print("Senha: ");
        String senha = reader.readLine();
        out.writeObject(login);
        out.writeObject(senha);
        out.writeObject("Mensagem do servidor para o cliente.");
        out.flush();
        JFrame frame = new JFrame( title: "Mensagens do Servidor");
        JTextArea textArea = new JTextArea( rows: 20,  columns: 50);
        textArea.setEditable(false);
        frame.add(new JScrollPane(textArea));
        frame.pack();
        frame.setDefaultCloseOperation(3);
```

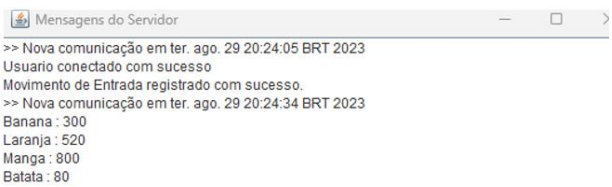
```
        frame.setVisible(true);
        ThreadClient threadClient = new ThreadClient(in, textArea);
        threadClient.start();

        while(menu) {
            System.out.println("L - Listar | E - Entrada | S - Saida | X - Finalizar");
            String opcao = reader.readLine().toUpperCase();
            out.writeObject(opcao);
            out.flush();
            switch (opcao) {
                case "X":
                    menu = false;
                case "L":
                    break;
                case "E":
                case "S":
                    System.out.print("ID da pessoa: ");
                    int idPessoa = Integer.parseInt(reader.readLine());
                    System.out.print("ID do produto: ");
                    int idProduto = Integer.parseInt(reader.readLine());
                    System.out.print("Quantidade: ");
                    int quantidade = Integer.parseInt(reader.readLine());
                    System.out.print("Valor unitário: ");
                    float valorUnitario = Float.parseFloat(reader.readLine());
                    out.writeInt(idPessoa);
                    out.writeInt(idProduto);
                    out.writeInt(quantidade);
                    out.writeFloat(valorUnitario);
                    out.flush();
                    break;
                default:
                    System.out.println("Opção inválida.");
            }
        }

    }
}
```

ThreadClient:

```
package cadastroclientv2;

import ...

no usages
public class ThreadClient extends Thread {
    no usages
    private ObjectInputStream in;
    no usages
    private final JTextArea textArea;

    no usages
    public ThreadClient(ObjectInputStream in, JTextArea textArea) {
        this.in = in;
        this.textArea = textArea;
    }

    public void run() {
        try {
            while(true) {
                Object data = this.in.readObject();
                String mensagem = (String)data;
                SwingUtilities.invokeLater(() -> {
                    this.textArea.append(mensagem + "\n");
                    this.textArea.setCaretPosition(this.textArea.getDocument().getLength());
                });
            }
        } catch (Exception var3) {
        }
    }
}
```

Resultado da execução:





a) Como as Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor?

   Por meio de uma Thread designada à comunicação com o servidor, o cliente tem a capacidade de continuar sua execução normal enquanto aguarda por respostas. Isso preserva a responsividade da interface gráfica, prevenindo possíveis bloqueios. Quando as respostas são recebidas, a Thread assíncrona encarregada atualiza a interface do usuário. Esse processo possibilita uma experiência mais agradável para o usuário, já que as operações de comunicação não interferem na interatividade da interface.

b) Para que serve o método invokeLater, da classe SwingUtilities?

   O método `invokeLater` da classe `SwingUtilities` é empregado para executar uma ação específica de maneira assíncrona na thread de eventos do Swing. Essa thread é encarregada de atualizar a interface gráfica. Essa abordagem é fundamental para garantir que as atualizações na interface ocorram de maneira segura, prevenindo conflitos entre threads e preservando a capacidade de resposta da aplicação.

c) Como os objetos são enviados e recebidos pelo Socket Java?

   Em Java, a transferência de objetos entre sistemas via sockets é realizada por meio do processo de serialização. A serialização converte esses objetos em uma sequência de bytes que podem ser transmitidos pela rede. O `ObjectOutputStream` é empregado para escrever os objetos como bytes, que são então enviados por meio do socket. Em contrapartida, o `ObjectInputStream` é utilizado para ler os bytes recebidos e reconstruir os objetos em sua forma original. É importante observar que a classe dos objetos deve implementar a interface `Serializable` para permitir tanto a serialização quanto a desserialização desses objetos.

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

Ao utilizar sockets em Java, a abordagem assíncrona permite que os clientes continuem executando outras tarefas enquanto aguardam respostas do servidor. Isso previne bloqueios e mantém a responsividade do programa. Por contraste, a abordagem síncrona exige que o cliente espere pela resposta do servidor antes de prosseguir, o que pode resultar em bloqueios e tornar o programa menos eficiente em termos de utilização de recursos e tempo de resposta. A preferência pelo comportamento assíncrono se dá para assegurar uma interação fluida do usuário e otimizar a eficiência na utilização dos recursos do sistema.

Conclusão

Desenvolvi uma aplicação cliente-servidor para gerenciamento de cadastros. Implementei de forma robusta as funcionalidades de autenticação, entradas e saídas, organizei eficientemente o código com classes de controle separadas, utilizei threads de forma adequada para comunicação assíncrona e criei a interface gráfica com a biblioteca Swing. Isso proporcionou-me uma valiosa experiência na aplicação de conceitos teóricos em um cenário prático.