Estácio

Campus: POLO SAGUAÇU - JOINVILLE - SC

Curso: DESENVOLVIMENTO FULL STACK

Disciplina: RPG0018 - Por que não paralelizar

Turma: 9001

Semestre: 1° Semestre (2024)

Aluno: Jederson Borges de Oliveira

Link: https://github.com/JedersonBorges/CadastroServer.git

Por que não paralelizar

2° Procedimento

Objetivos 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.**
5. **No final do exercício, o aluno terá criado um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java para implementação de clientes síncronos e assíncronos. As Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assíncrona.**

Código utilizado

CadastroClient

```
package com.cadastro;

import java.io.*;
import java.net.*;

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())) {
```

```
            System.out.println("Conectado ao servidor...");

            out.writeObject("op1");
            out.writeObject("op1");
            out.flush();

            String response = (String) in.readObject();
            System.out.println(response);

            if ("Usuário conectado com sucesso".equals(response)) {

                out.writeObject("L");
                out.flush();


                Object produto;
                while ((produto = in.readObject()) != null) {
                    System.out.println(produto);
                }
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

CadastroServer:

```
package com.cadastro;

import java.io.*;
import java.net.*;
import java.util.*;

public class CadastroServer {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(4321)) {
            System.out.println("Servidor iniciado na porta 4321...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Cliente conectado!");

                new CadastroThread(clientSocket).start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

class CadastroThread extends Thread {
    private Socket clientSocket;

    public CadastroThread(Socket socket) {
```

```
        this.clientSocket = socket;
    }

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

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

            if (validarCredenciais(login, senha)) {
                out.writeObject("Usuário conectado com sucesso");
                out.flush();

                String comando = (String) in.readObject();
                if ("L".equals(comando)) {
                    List<String> produtos = Arrays.asList("Banana",
"Laranja", "Manga");
                    for (String produto : produtos) {
                        out.writeObject(produto);
                        out.flush();
                    }
                }
            } else {
                out.writeObject("Credenciais inválidas");
                out.flush();
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    private boolean validarCredenciais(String login, String senha) {
        return "op1".equals(login) && "op1".equals(senha);
    }
}
```

- build.xml

```
<project name="CadastroServer" default="compile" basedir=".">
    <target name="compile">
        <mkdir dir="build"/>
        <javac srcdir="src" destdir="build" includeantruntime="false"/>
    </target>

    <target name="run" depends="compile">
        <java classname="com.cadastro.CadastroServer" classpath="build">
            <classpath>
                <pathelement path="lib/*"/>
            </classpath>
        </java>
    </target>
</project>
```

Análise e Conclusão

Como funcionam as classes Socket e ServerSocket?

O ServerSocket é o que o servidor usa para ouvir e esperar conexões dos clientes em uma porta específica, quando um cliente tenta se conectar, o servidor aceita a conexão e cria um Socket para se comunicar com o cliente

O Socket é utilizado tanto pelo servidor quanto pelo cliente, para enviar e receber dados depois que a conexão foi estabelecida. O cliente cria um Socket para se conectar ao servidor na porta e endereço desejados, após isso eles podem trocar mensagens.

Qual a importância das portas para a conexão com servidores?

As portas são fundamentais para a conexão com servidores porque elas funcionam como canais de comunicação específicos

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

As classes ObjectInputStream e ObjectOutputStream são usadas para ler e escrever objetos em Java, facilitando a comunicação entre diferentes partes de um programa ou entre diferentes programas.

Os objetos transmitidos precisam ser serializáveis porque isso permite que eles sejam convertidos em um formato que pode ser facilmente transmitido através de uma rede ou salvo em um arquivo.

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

Mesmo com as classes de entidades JPA no cliente, o acesso ao banco de dados é isolado porque o cliente não interage diretamente com o banco.