Disciplina: Nível 5 - Por Que Não Paralelizar
Semestre: 1º - 2024
Aluno: DAVIDSON PEREIRA DE OLIVEIRA
Matrícula: 2022.11.54585-5
Repositório no GIT: dev-davidson/RPG0018-Por-que-n-o-paralelizar (github.com)

Relatório discente de acompanhamento

👉 1º Procedimento | Criando o Servidor e Cliente de Teste

Códigos:

Servidor (CadastroServer):

```
import java.io.*;
import java.net.*;
import java.util.List;
import model.Produto;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;

public class CadastroThread extends Thread {
    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private Socket s1;

    public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, Socket s1) {
        this.ctrl = ctrl;
        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
    }

    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();
```

```
            if (ctrlUsu.findUsuario(login, senha) == null) {
                s1.close();
                return;
            }


            while (true) {

                String comando = (String) in.readObject();


                if (comando.equals("L")) {
                    List<Produto> produtos = ctrl.findProdutoEntities();
                    out.writeObject(produtos);
                }
            }
        } catch (IOException | ClassNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import controller.ProdutoJpaController;
```

```
import controller.UsuarioJpaController;

public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
        ProdutoJpaController ctrl = new ProdutoJpaController(emf);
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

        try (ServerSocket ss = new ServerSocket(4321)) {
            while (true) {
                Socket s = ss.accept();
                CadastroThread ct = new CadastroThread(ctrl, ctrlUsu, s);
                ct.start();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Cliente (CadastroClient):

```
import java.io.*;
import java.net.*;
import java.util.List;
import model.Produto;

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());


            out.writeObject("op1"); // exemplo login
            out.writeObject("op1"); // exemplo senha


            out.writeObject("L");


            List<Produto> produtos = (List<Produto>) in.readObject();
            for (Produto p : produtos) {
                System.out.println(p.getNome());
            }
        } catch (IOException | ClassNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}
```

Análise e Conclusão:

a) Como funcionam as classes Socket e ServerSocket?

- A classe ServerSocket é usada no lado do servidor para aguardar e aceitar solicitações de conexão dos clientes. Ele está constantemente ouvindo em uma porta específica para conexões entrantes. Quando uma conexão é aceita, ele retorna um objeto Socket que representa a conexão entre o cliente e o servidor.

- A classe Socket é usada tanto no lado do cliente quanto no lado do servidor para estabelecer uma conexão entre os dois. Ela fornece os fluxos de entrada e saída para enviar e receber dados entre o cliente e o servidor.

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

As portas são importantes porque ajudam na identificação de processos e serviços em um sistema de rede. Em uma conexão TCP/IP, por exemplo, a porta é um número de 16 bits que identifica um processo específico em um host. No contexto de conexão com servidores, as portas são usadas para direcionar as solicitações de conexão para os serviços corretos que estão sendo executados em um servidor.

c) 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 a entrada e saída de objetos Java serializados. Elas permitem que objetos Java completos sejam transmitidos através de uma conexão de rede.

- Os objetos transmitidos devem ser serializáveis para que possam ser convertidos em uma sequência de bytes que podem ser enviados pela rede e reconstruídos no lado receptor. A serialização é o processo de converter um objeto em uma sequência de bytes e a desserialização é o processo de reconstruir o objeto a partir desses bytes.

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

Mesmo que as classes de entidades JPA estejam presentes no cliente, o acesso real ao banco de dados não ocorre no cliente, mas sim no servidor. O cliente envia solicitações para o servidor e o servidor manipula os dados do banco de dados de acordo com essas solicitações. Portanto, o acesso ao banco de dados permanece isolado no servidor, garantindo que todas as operações de acesso aos dados sejam controladas centralmente e que o cliente não tenha acesso direto ao banco de dados.