



Desenvolvimento Full Stack

Aluno: Victor de A. Costa

Missão Prática | Nível 5 | Mundo 3 - 2024.3
POR QUE NÃO PARALELIZAR!

Servidores e clientes baseados em Socket, com uso de Threads tanto no lado cliente quanto no lado servidor, acessando o banco de dados via JPA.

1º Procedimento | Criando o Servidor e Cliente de Teste

---> CÓDIGOS UTILIZADOS

```
CadastroClient.java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
 * license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package cadastroclient;
/**
 *
 *
 * @author victorcosta
 */
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.io.StreamCorruptedException;
import java.io.InvalidClassException;
public class CadastroClient {
public static void main(String[] args) {
try {
// Instanciar um Socket apontando para localhost, na porta 4321
Socket socket = new Socket("localhost", 4321);
// Encapsular os canais de entrada e saída do Socket em objetos dos tipos
// ObjectOutputStream (saída) e ObjectInputStream (entrada)
ObjectOutputStream output = new ObjectOutputStream(socket.getOutputStream());
ObjectInputStream input = new ObjectInputStream(socket.getInputStream());
// Escrever o login e a senha na saída, utilizando os dados de algum dos registros
// da tabela de usuários (op1/op1)
String login = "op1";
String senha = "op1";
output.writeObject(login);
output.writeObject(senha);
output.writeObject("L");
// Receber a coleção de entidades no canal de entrada
System.out.println("Usuario conectado com sucesso");
try {
String[] produtoNames = (String[]) input.readObject();
for (String nome : produtoNames) {
System.out.println(nome);
}
} catch (StreamCorruptedException e) {
System.err.println("Stream corrupted: " + e.getMessage());
} catch (InvalidClassException e) {
System.err.println("Invalid class: " + e.getMessage());
} catch (IOException e) {
System.err.println("IO error: " + e.getMessage());
}
socket.close();
} catch (IOException e) {
System.err.println("Erro ao conectar ao servidor: " + e.getMessage());
} catch (ClassNotFoundException e) {
System.err.println("Erro ao ler objeto: " + e.getMessage());
}
}
}
```

```

CadastroServer.java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
 * license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
 */
package cadastroserver;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.net.ServerSocket;
import java.net.Socket;
import java.io.IOException;
public class CadastroServer {
    public static void main(String[] args) {
        // Instanciar um objeto do tipo EntityManagerFactory a partir da unidade de persistência
        EntityManagerFactory emf =
            Persistence.createEntityManagerFactory("CadastroServerPU");

        // Instanciar o objeto ctrl, do tipo ProdutoJpaController
        ProdutoJpaController ctrl = new ProdutoJpaController(emf);

        // Instanciar o objeto ctrlUsu do tipo UsuarioJpaController
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
        ServerSocket serverSocket = null;

        // Instanciar um objeto do tipo ServerSocket, escutando a porta 4321

        try {
            serverSocket = new ServerSocket(4321);
        } catch (IOException e) {
            System.err.println("Error Server: " + e.getMessage());
        }
        System.out.println("Servidor iniciado. Aguardando conexões...");
        while (true) {
            // Obter a requisição de conexão do cliente
            Socket socket = null;
            try {
                socket = serverSocket.accept();
            } catch (IOException e) {
                System.err.println("Error Socket: " + e.getMessage());
            }
            System.out.println("Nova conexão estabelecida.");
            // Instanciar uma Thread, com a passagem de ctrl, ctrlUsu e do Socket da conexão
            CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, socket);
            // Iniciar a Thread
            thread.start();
        }
    }
}

CadastroThread.java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
 * license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package cadastroserver;
/**
 *
 * @author victorcosta
 */
import model.Usuario;

import java.util.*;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import model.Produto;

```

```
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 {
            out = new ObjectOutputStream(s1.getOutputStream());
            in = new ObjectInputStream(s1.getInputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void run() {
        try {
            String login = (String) in.readObject();
            String senha = (String) in.readObject();
            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                s1.close();
                return;
            }
            while (true) {
                String comando = (String) in.readObject();
                if (comando.equals("L")) {

                    List<Produto> produtos = ctrl.findAll();
                    Produto[] produtoArray = produtos.toArray(new Produto[produtos.size()]);
                    String[] produtoNames = new String[produtoArray.length];
                    for (int i = 0; i < produtoArray.length; i++) {
                        produtoNames[i] = produtoArray[i].getNome();
                    }
                    out.writeObject(produtoNames);
                }
            }
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Fim da conexão");
        } finally {
            try {
                s1.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
*/
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
 * license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package controller;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import javax.persistence.NoResultException;
import model.Usuario;
/**
 *
 * @author victorcosta
 */
public class UsuarioJpaController {
    private EntityManager em;
    public UsuarioJpaController(EntityManagerFactory emf) {
        this.em = emf.createEntityManager();
    }
}
```

```

public Usuario findUsuario(String login, String senha) {
    try {
        Query query = em.createNamedQuery("Usuario.findByLoginSenha");
        query.setParameter("login", login);
        query.setParameter("senha", senha);
        return (Usuario) query.getSingleResult();
    } catch (NoResultException e) {
        return null;
    }
}
}

ProdutoJpaController.java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
 * license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package controller;
/**
 *
 * @author victorcosta
 */
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import java.util.List;
import model.Produto;
public class ProdutoJpaController {
    private EntityManagerFactory emf;
    public ProdutoJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }
    public List<Produto> findAll() {
        EntityManager em = emf.createEntityManager();
        try {
            return em.createNamedQuery("Produto.findAll").getResultList();
        } finally {
            em.close();
        }
    }
}

```

---> RESULTADO

The screenshot shows the Eclipse IDE interface with the following components:

- Output View:** Displays the execution logs for two running processes: "CadastroServer (run)" and "CadastroClient (run)". The logs show the connection process and some test data being inserted.
- Database View:** A table titled "Results" showing the data inserted into the database. The table has columns: idProduto, nome, quantidade, and precoVenda.
- Status Bar:** Shows the message "Query executed successfully." at the bottom.

	idProduto	nome	quantidade	precoVenda
1	1	nome produto	50	5000
2	2	213213	50	5000
3	3	Camiseta Azul	564	7000
4	4	Calça jeans	15	400
5	12	2132	213	12321
6	14	sqdas	2	134
7	15	adsdas	43	32423
8	16	adsdas	43	32423
9	17	21312	213	21312
10	18	21312	213	21312
11	23	test	324	234
12	26	novo	21312	321321

---> ANÁLISE E CONCLUSÃO

A. Como funcionam as classes Socket e ServerSocket?

A classe Socket representa uma conexão de rede entre um cliente e um servidor. Ela é usada quando um aplicativo deseja estabelecer uma conexão com um serviço remoto, já a classe ServerSocket é utilizada para criar um ouvinte de conexões em uma porta específica. Ela permite que o servidor aceite conexões de clientes.

B. Qual a importância das portas para a conexão com servidores?

As portas são cruciais para a comunicação entre clientes e servidores em redes. Elas funcionam como pontos de acesso que identificam serviços específicos em um servidor, permitindo que múltiplos serviços rodem simultaneamente em uma única máquina sem interferência.

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

A classe ObjectInputStream é utilizada para ler objetos que foram previamente gravados em uma stream. Já a classe ObjectOutputStream é utilizada para gravar objetos em uma stream, convertendo objetos Java em um fluxo de bytes que pode ser armazenado em um arquivo ou transmitido através de uma rede.

Os objetos devem ser serializáveis, para poderem ser convertidos em um fluxo de bytes para armazenamento ou transmissão. A serialização permite que os objetos sejam persistidos ou enviados pela rede e depois reconstruídos em seu estado original.

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

Ao utilizar as classes de entidades JPA no cliente, o isolamento do acesso ao banco de dados é garantido por alguns motivos principais: Separação de Camadas, Transparência de Acesso, Controle Transacional, Segurança, Facilidade de Manutenção.