POLO OLARIA - RIO DE JANEIRO - RJ

DESENVOLVIMENTO FULL STACK

Nivel 5: Por Que Não Paralelizar? | Turma 9001

Erik Bastos de Moraes

Missão Prática | Nível 5 | Mundo 3

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

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.

UsuarioJpaController.java

```
package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.TypedQuery;
import model.Usuario;

public class UsuarioJpaController {

    private EntityManagerFactory emf = null;

    public UsuarioJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public Usuario findUsuario(String login, String senha) {
        EntityManager em = getEntityManager();
        try {
            TypedQuery<Usuario> query = em.createQuery(
                "SELECT u FROM Usuario u WHERE u.login = :login AND u.senha = :senha",
Usuario.class);
            query.setParameter("login", login);
            query.setParameter("senha", senha);
            return query.getResultStream().findFirst().orElse(null);
        } finally {
```

```
            em.close();
        }
    }
}
```

ProdutoJpaController.java

```
package controller;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import java.util.List;
import model.Produto;

public class ProdutoJpaController {

    private EntityManagerFactory emf = null;

    public ProdutoJpaController(EntityManagerFactory emf) {
        this.emf = emf;
    }

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public List<Produto> findProdutoEntities() {
        EntityManager em = getEntityManager();
        try {
            return em.createQuery("SELECT p FROM Produto p",
Produto.class).getResultList();
        } finally {
            em.close();
        }
    }
}
```

CadastroServer.java

```
package cadastroserver;


import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import model.Produto;
import model.Usuario;
import java.util.List;


public class CadastroServer extends Thread {


    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private Socket s1;


    public CadastroServer(ProdutoJpaController ctrl, UsuarioJpaController 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();

            Usuario u = ctrlUsu.findUsuario(login, senha);
            if (u == null) {
                out.writeObject("LOGIN INVALIDO");
                s1.close();
                return;
            } else {
                out.writeObject("LOGIN OK");
            }

            while (true) {
                String comando = (String) in.readObject();
                if ("L".equalsIgnoreCase(comando)) {
                    List<Produto> produtos = ctrl.findProdutoEntities();
                    out.writeObject(produtos);
                } else {
                    break;
                }
            }

            s1.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Main.java

```
package cadastroserver;


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


public class Main {
    public static void main(String[] args) {
        try {
            System.out.println("Iniciando servidor...");


            EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroPU");


            ProdutoJpaController ctrl = new ProdutoJpaController(emf);
            UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);


            ServerSocket server = new ServerSocket(4321);
            System.out.println("Servidor escutando na porta 4321...");
```

```
            while (true) {
                try {
                    Socket s = server.accept();
                    System.out.println("Cliente conectado: " +
s.getInetAddress().getHostAddress());

                    CadastroServer t = new CadastroServer(ctrl, ctrlUsu, s);
                    t.start();
                } catch (EOFException eof) {
                    System.out.println("Conexão encerrada pelo cliente.");
                } catch (IOException e) {
                    System.err.println("Erro ao aceitar conexão: " + e.getMessage());
                    e.printStackTrace();
                }
            }

        } catch (Exception e) {
            System.err.println("Erro ao iniciar o servidor: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

CadastroClient.java

```
package cadastroclient;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
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");
            out.writeObject("op1");

            String respostaLogin = (String) in.readObject();
            System.out.println("Resposta do servidor: " + respostaLogin);

            if ("LOGIN OK".equals(respostaLogin)) {
                out.writeObject("L");

                List<Produto> produtos = (List<Produto>) in.readObject();
                System.out.println("Produtos recebidos:");
                for (Produto p : produtos) {
```

```
                System.out.println("- " + p.getNome()); // ajusta de acordo com sua entidade
            }
        }

        socket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Resultado

run:

Resposta do servidor: LOGIN OK

Produtos recebidos:

- PortalGun

- DragonSlayer

- Pokebola

BUILD SUCCESSFUL (total time: 1 second)

Análise e Conclusão

A. Como funcionam as classes Socket e ServerSocket?

R: O ServerSocket se conecta a uma porta e aguarda novas conexões de cliente TCP. Quando uma nova conexão de cliente TCP é recebida, uma instância da classe Socket é criada pela instância ServerSocket e usada para se comunicar com o cliente remoto.

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

R: As portas de rede atuam como um sistema de identificação de serviços, direcionando as solicitações de rede para o aplicativo ou processo correto em um computador. Sem elas, um servidor não conseguiria discernir entre diferentes tipos de tráfego e não poderia fornecer os serviços adequados.

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

R: São utilizadas em Java para realizar serialização e desserialização de objetos, respetivamente. ObjectOutputStream converte um objeto em um fluxo de bytes, enquanto ObjectInputStream reconstruí-lo de um fluxo de bytes. Para que este processo funcione, os objetos precisam ser serializáveis, então precisam implementar a interface Serializable.

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

R: Porque somente o servidor realiza efetivamente a comunicação com o banco de dados, as entidades JPA no cliente são apenas estruturais.

---

https://github.com/ErikBM2661/CadastroServer.git