RPG0018 - Por que não paralelizar

Kervini Ribeiro da Silva - 202301206073

Polo Centro – Araraquara – SP
Por Que Não Paralelizar? – 9001 – 3º Semestre

Objetivo da Prática

Criar sistema servidor e cliente trocando mensagens via socket, utilizando threads para processamento paralelo e acesso a banco via JPA.

GITHUB: https://github.com/Kervini/Mundo3-Nivel5.git

1º Procedimento | Criando o Servidor e Cliente de Teste

Análise e Conclusão:

a) Como funcionam as classes Socket e ServerSocket? **R:** São objetos que encapsulam uma conexão entre dois sistemas, onde por parâmetro enviamos a porta de comunicação. Assim que a conexão é aberta ela é mantida até que a troca de mensagens é finalizada.

b) Qual a importância das portas para a conexão com servidores? **R:** As portas são uma forma de abrir um meio de comunicação entre dois componentes em uma rede e sendo reservada unicamente para quem esse propósito. Com as portas a conexão é realizada e mantida, onde os dois lados podem trocar informações e isso é um recurso muito importante para comunicação de sistemas.

c) Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis? **R:** As classes de entrada e saida de objetos, como o próprio nome já diz é uma forma de envio de objetos do java, elas nos permitem transmitir objetos pelo conal de conexão e esses objetos para possam ser transmitidos precisam ser serializados em bits e depois reconstruídos no remetente.

d) Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados? **R:** O framework JPA em conjunto com as classes Entity abstraem totalmente a manipulação de dados no banco de dados, exigindo pouca ou nenhuma atenção do programador.

```java
package cadastroserver;

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

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

        try {
            ServerSocket serverSocket = new ServerSocket(4321);

            while(true){
                Socket clientSocket = serverSocket.accept();

                Thread thread = new Thread(new CadastroThread(
                        clientSocket, ctrlProd, ctrlUsu));
                thread.start();
            }

        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
    }

}
```

```java
public class CadastroThread implements Runnable {
    private final Socket sl;
    private final ProdutoJpaController ctrl;
    private final UsuarioJpaController ctrlUsu;

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

    @Override
    public void run() {
        try {
            ObjectOutputStream out =
                    new ObjectOutputStream(sl.getOutputStream());
            BufferedReader in = new BufferedReader(
                    new InputStreamReader(sl.getInputStream()));

            String login = in.readLine();
            String senha = in.readLine();
            Usuario usuario = this.ctrlUsu.getUsuario(login, senha);

            if (usuario == null) {
                sl.close();
                return;
            }

            while (true) {
                String comando = in.readLine().toUpperCase();
                if (comando == null)
                    continue;

                if (comando.equals("L")) {
                    List<Produto> produtos = ctrl.findAll();
                    out.writeObject(produtos);
                }
            }

        } catch (IOException ex) {
            System.out.println("Erro de conexão: " + ex.getMessage());
        }
    }
}
```

```java
package controller;

import java.io.Serializable;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
import model.Usuario;

public class UsuarioJpaController implements Serializable {
    private EntityManagerFactory emf = null;

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

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

    public Usuario getUsuario(String login, String senha){
        EntityManager em = getEntityManager();
        Query q = em.createNamedQuery("Usuario.findUsuario");
        q.setParameter("login", login);
        q.setParameter("senha", senha);
        return (Usuario) q.getSingleResult();
    }
}
```

```java
package controller;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
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> findAll(){
        EntityManager em = getEntityManager();
        Query q = em.createNamedQuery("Produto.findAll");

        return q.getResultList();
    }
}
```

```java
import java.io.ObjectInputStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.List;
import java.util.Scanner;
import model.Produto;

public class CadastroClient {
    public static void main(String[] args) {
        try {
            Scanner entrada = new Scanner(System.in);
            Socket socket = new Socket("localhost", 4321);

            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            System.out.println("Digite o login: ");
            out.println(entrada.nextLine());
            System.out.println("Digite a senha: ");
            out.println(entrada.nextLine());
            System.out.println("Faz o L: ");
            out.println(entrada.nextLine());

            ObjectInputStream in =
                    new ObjectInputStream(socket.getInputStream());
            List<Produto> produtos = (List<Produto>)in.readObject();

            for(Produto p : produtos){
                System.out.println(p.getNome());
            }

            in.close();
            out.close();
            socket.close();

        } catch (IOException ex) {
            System.out.println("Erro de conexão com o servidor: "
                    + ex.getMessage());
        } catch (ClassNotFoundException ex) {
            System.out.println(ex.getMessage());
        }
    }

}
```

2º Procedimento | Servidor Completo e Cliente Assíncrono

Análise e Conclusão:

a) Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados? **R:** O tratamento das respostas do servidor se torna assíncrono devido a thread que roda em paralelo com a aplicação principal e fica ouvindo o canal de entrada de dados (InputObjectStream). Enquanto o programa principal envia comandos a a thread recebe as respostas e isso caracteriza um comportamento assíncrono.

b) Como o uso de operador *lambda* simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java? **R:** O método InvokeLater trabalha com a thread do Swing, ele garante a atualização da interface assim que todos os recursos forem utilizados.

c) Por que métodos acionados diretamente pelo método main, sem o uso de um objeto, precisam ser marcados como *static*? **R:** Os objetos são recebidos em formato binário atras do objeto de comunicação ObjectInputStream.

d) Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento. **R:** Na primeira versão utilizando o comportamento síncrono, o cliente só envia ou recebia mensagens do servidor, ou seja, era algo sequencial. Já na segunda versão como assíncrono o cliente podia de forma paralela enviar e receber mensagens do servidor. Uma thread principal responsável por enviar mensagens e uma thread em paralelo que ficava esperando respostas do servidor e processava essas respostas.

```
18
19  public class CadastroThreadV2 implements Runnable{
20
21      private final Socket sl;
22      private final ProdutoJpaController ctrlProd;
23      private final UsuarioJpaController ctrlUsu;
24      private final MovimentoJpaController ctrlMov;
25      private final PessoaJpaController ctrlPessoa;
26
27      public CadastroThreadV2(Socket sl, ProdutoJpaController ctrlProd,
28              UsuarioJpaController ctrlUsu, MovimentoJpaController ctrlMov,
29              PessoaJpaController ctrlPessoa) {
30          this.sl = sl;
31          this.ctrlProd = ctrlProd;
32          this.ctrlUsu = ctrlUsu;
33          this.ctrlMov = ctrlMov;
34          this.ctrlPessoa = ctrlPessoa;
35      }
36
37      @Override
38      public void run() {
39          try {
40              ObjectOutputStream out = new ObjectOutputStream(
41                      sl.getOutputStream());
42              ObjectInputStream in = new ObjectInputStream(
43                      sl.getInputStream());
44              Scanner entrada = new Scanner(System.in);
45
46              String login = in.readObject().toString();
47              String senha = in.readObject().toString();
48
49              Usuario usuario = this.ctrlUsu.getUsuario(login, senha);
50
51              if (usuario == null) {
52                  sl.close();
53                  return;
54              }
55
56              System.out.println("Acesso permitido!");
57
```

```
58              while (true) {
59                  String comando = in.readObject().toString().toUpperCase();
60                  if (comando == null)
61                      continue;
62
63                  if (comando.equals("L")) {
64                      List<Produto> produtos = ctrlProd.findAll();
65                      out.writeObject(produtos);
66                  }else if(comando.equals("S") || comando.equals("E")){
67                      Movimento movimento = new Movimento();
68                      movimento.setIdusuario(usuario);
69                      movimento.setTipoMovimento(comando.toLowerCase().charAt(0));
70
71                      String idPessoa = in.readObject().toString();
72                      movimento.setIdpessoa(ctrlPessoa.findById(
73                              Integer.parseInt(idPessoa)));
74
75                      String idProduto = in.readObject().toString();
76                      Produto produto = ctrlProd.findById(
77                              Integer.parseInt(idProduto));
78                      movimento.setIdproduto(produto);
79
80                      movimento.setQuantidade(Integer.parseInt(
81                              in.readObject().toString()));
82
83                      if(comando.equals("E"))
84                          movimento.setValorUnitario(Float.parseFloat(
85                                  in.readObject().toString()));
86                      else
87                          movimento.setValorUnitario(produto.getPrecoVenda());
88
89                      ctrlMov.persist(movimento);
90
91                      produto.atualizaQuantidade(
92                              movimento.getQuantidade(), comando.equals("E"));
93                      ctrlProd.merge(produto);
94
95                      out.writeObject("Movimento realizado!");
96                  }
97              }
98
```

```java
public class CadastroClientV2 {

    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());

            InputStreamReader reader = new InputStreamReader(System.in);
            BufferedReader entrada = new BufferedReader(reader);

            System.out.println("Conectado...");

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

            SaidaFrame janela = new SaidaFrame();

            Thread thread = new Thread( new ThreadClient(in, janela, socket));
            thread.start();
            String opcao;

            do {
                System.out.println("\n----------------------");
                System.out.println("L - Listar |"
                        + " X - Finalizar | E - Entrada | S - Saida");
                opcao = entrada.readLine().toUpperCase();

                switch (opcao) {
                    case "L" -> {
                        out.writeObject(opcao);
                    }
                    case "X" -> {
                        System.out.println("Finalizando...");
                    }
```

```
                    case "E" -> {
                        out.writeObject(opcao);
                        System.out.println("Digite o ID da pessoa: ");
                        out.writeObject(entrada.readLine());
                        System.out.println("Digite o ID do produto: ");
                        out.writeObject(entrada.readLine());
                        System.out.println("Digite a quantidade: ");
                        out.writeObject(entrada.readLine());
                        System.out.println("Digite o valor unitario: ");
                        out.writeObject(entrada.readLine());
                    }
                    case "S" -> {
                        out.writeObject(opcao);
                        System.out.println("Digite o ID da pessoa: ");
                        out.writeObject(entrada.readLine());
                        System.out.println("Digite o ID do produto: ");
                        out.writeObject(entrada.readLine());
                        System.out.println("Digite a quantidade: ");
                        out.writeObject(entrada.readLine());
                    }
                    default -> {
                        System.out.println("Opcao invalida!");
                    }
                }
            } while (!opcao.equals("X"));
            in.close();
            out.close();
            socket.close();
        } catch (IOException ex) {
            Logger.getLogger(ThreadClient.class.getName())
                    .log(Level.SEVERE, null, ex);
        }
    }
}
```

```java
public class ThreadClient implements Runnable {
    private ObjectInputStream entrada;
    //private JTextArea textArea;
    private SaidaFrame janela;
    private Socket socket;

    public ThreadClient(ObjectInputStream entrada,
            SaidaFrame janela, Socket socket) {
        this.entrada = entrada;
        this.janela = janela;
        this.socket = socket;
    }

    @Override
    public void run() {
        while (true) {
            try {
                String texto = "\n";
                Object resposta = entrada.readObject();

                if (resposta instanceof Collection) {
                    List<Produto> produtos = (List<Produto>) resposta;
                    texto += "\nLista de Produtos";
                    for (Produto p : produtos) {
                        texto += "\nProduto: " + p.getNome() +
                                "; Quantidade: " + p.getQuantidade();
                    }
                } else {
                    texto += "\n" + resposta.toString();
                }

                final String msg = texto;
                SwingUtilities.invokeLater(() -> {
                    janela.texto.append(msg);
                });
            } catch (IOException ex) {
                Logger.getLogger(ThreadClient.class.getName())
                        .log(Level.SEVERE, null, ex);
            } catch (ClassNotFoundException ex) {
                Logger.getLogger(ThreadClient.class.getName())
                        .log(Level.SEVERE, null, ex);
            }
        }
    }
```

```java
package cadastroclientv2;

import javax.swing.JDialog;
import javax.swing.JTextArea;

public class SaidaFrame extends JDialog{
    public JTextArea texto;

    public SaidaFrame(){
        this.setBounds(100, 100, 400, 400);
        this.setModal(false);
        texto = new JTextArea();
        texto.setText("Usuario conectado!");
        texto.setEditable(false);
        this.add(texto);
        this.setVisible(true);
    }
}

```

Conclusão

A missão foi muito importante para explorar o uso de threads e nos proporcionar essa experiência de desenvolver dois sistemas trocando mensagens de forma assíncrona.