Sistemas Distribuídos e Paralelos - Trabalho Prático: Parte 3

Daniel Fernandes Pinho - 2634 [1]
Taianne Valerie - 2679[1]

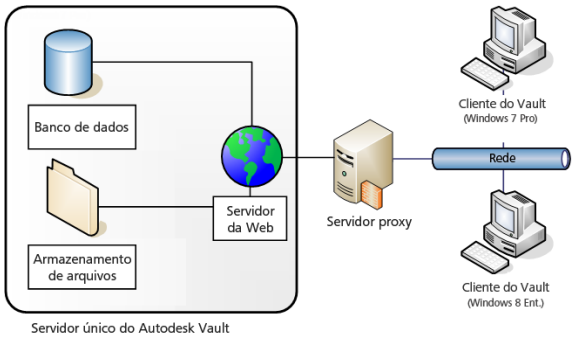
[1]Instituto de Ciências Exatas e Tecnológicas,
Universidade Federal de Viçosa, Florestal, MG, Brasil

(daniel.f.pinho)[1], (taianne.mota)[1]@ufv.br

1. Introdução

Neste trabalho foi desenvolvido um sistema distribuído utilizando API de Sockets. Para seu funcionamento, é necessário uma cominucação entre processos com trocas de mensagens. Para que essas trocas ocorressem, utilizamos persistência em Banco de Dados, com o connector J (JDBC) para uso com SQLite.

Para que o fluxo das mensagens ocorram, o Servidor é inicializado em *localhost* e espera por conexões de novas instâncias de cliente. Enquanto isso, o Cliente é inicializado e tenta conectar ao Servidor mandando mensagens codificadas. O Servidor descodifica essas mensagens e gerencia o banco de dados de acordo com as operações escolhidas pelo Cliente, gerando uma resposta.s



2. Principais decisões

1. Dentre as decisões que precisamos tomar, a maior delas foi decidir qual banco de dados utilizar para fazer a conexão em Java. Inicialmente pensamos em utilizar o MYSQL, mas com as complicações durante o desenvolvimento optamos por seguir com SQLite, que trouxe uma maior facilidade de código e execução.
2. Outra decisão foi utilizar a LP Java. Essa decisão foi tomada por termos uma maior intimidade com a linguagem de programação.

2.1. Conexão

A conexão aconteceu pelo conector J. Para que isso fosse possível, baixamos o Driver de SQLite e utilizamos sua biblioteca no projeto. A persistência foi feita através de Banco de Dados usando um Banco de Dados mais simples: SQLite.

colocar aqui uma imagem

Para a conexão, apenas duas linhas de código foi necessária:

```
 String jdbcUrl = "jdbc:sqlite:C:\\Users\\danie\\Documents\\Dist2\\BD\\
    db_OTF2020.db";
Connection conn = DriverManager.getConnection(jdbcUrl);
```

2.2. Códigos produzidos

2.2.1. Classe Cliente (Principal)

```
public class OTF2020_main {
    public static void main(String[] args) throws Exception {
            ClienteSockets clienteSocket = new ClienteSockets();
            clienteSocket.run();
    }
}
```

2.2.2. Classe ClienteSockets

```
package com.APISocketsClientes;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class ClienteSockets{

    private final int porta;

    public ClienteSockets(){
        this.porta = 3306;
    }

    public void run() {

        Socket socketObj = null;
        Scanner sc = new Scanner(System.in);

        try{
            socketObj = new Socket("localhost", this.porta);
            DataInputStream in = new DataInputStream(socketObj.
    getInputStream());
            DataOutputStream out = new DataOutputStream(socketObj.
    getOutputStream());
            while(true){
                String menu_mensagem = in.readUTF();
                System.out.println(menu_mensagem);
                System.out.println("Email");
                String email = sc.next();
                System.out.println("Senha");
```

```
                String senha = sc.next();
                out.writeUTF(email);
                out.writeUTF(senha);
                String resposta_servidor = in.readUTF();
                if(resposta_servidor.equals("Login Aprovado")){
                    out.writeUTF("continue");
                    in.readUTF();
                }
            }
        }catch (UnknownHostException e){
            System.out.println("Socket:"+e.getMessage());
        }catch (EOFException e){
            System.out.println("EOF:"+e.getMessage());
        }catch (IOException e){
            System.out.println("readline2:"+e.getMessage());
        }finally {
            if(socketObj!=null)
            try{
            socketObj.close();
        }catch (IOException e){
        System.out.println("Close:"+e.getMessage());}
        }
    }
}
```

2.2.3. Classe Servidor

```
public class Servidor {
    public static void main (String args[]) {
        try{
            int serverPort = 3306; // the server port
            ServerSocket listenSocket = new ServerSocket(serverPort);
            while(true) {
                new Conexao(listenSocket.accept()).start(); // Keep
    running
            }
        }catch(IOException e) {
            System.out.println("Listen socket:"+e.getMessage());
        }
    }
}
```

2.2.4. ServidorControle

```
package com.APISocketsServidor;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.PreparedStatement;

```

```java
public class ServidorControle{

    private String Email;
    private String Senha;

    public ServidorControle(String email, String senha){
        this.Email = email;
        this.Senha = senha;
    }

    public void printaMenu(){
        System.out.print("######-----Ol !! O que deseja fazer ?-----######\n\n");
        System.out.print("|-----------------------------------------------|\n");
        System.out.print("| Op  o 1 - Criar novo pacote de figurinhas |\n");
        System.out.print("| Op  o 2 - Criar novo  lbum  de figurinhas  |\n");
        System.out.print("| Op  o 3 - Realizar uma tarefa |\n");
        System.out.print("| Op  o 4 - Participar de um sorteio |\n");
        System.out.print("| Op  o 5 - Ver seus pontos |\n");
        System.out.print("| Op  o 6 - Acompanhar um pedido |\n");
        System.out.print("| Op  o 7 - Participar de um sorteio |\n");
        System.out.print("| Op  o 8 - Realizar trocas |\n");
        System.out.print("| Op  o 9 - Sair |\n");
        System.out.print("|-----------------------------------------------|\n");
    }

     // Metodos para gerencimento do banco de dados

    public void verificaAcesso(String email, String senha, Connection conn){
        try{
            String sql = "SELECT email, senha FROM Cadastro WHERE email = ? and senha = ?";
            PreparedStatement stmt  = conn.prepareStatement(sql);
            stmt.setString(1, email);
            stmt.setString(2, senha);
            stmt.execute(sql);
            //System.out.println(rs.getString("email") +  "\t" + rs.getString("senha"));
        }catch(SQLException e){
            System.out.println("Erro ao verificar acesso:"+e.getMessage());
        }
    }

```

```
49      public String getEmail(){
50          return this.Email;
51      }
52
53      public String getSenha(){
54          return this.Senha;
55      }
56  }
```

A implementação do servidor foi feita somente a parte de Conexão com a API Sockets, onde o servidor aguarda por novas conexões.

2.2.5. Classe Conexao

```
1   package com.APISocketsServidor;
2
3   import java.io.IOException;
4   import java.io.DataInputStream;
5   import java.io.DataOutputStream;
6   import java.net.Socket;
7   import java.util.logging.Level;
8   import java.util.logging.Logger;
9   import java.sql.DriverManager;
10  import java.sql.Connection;
11  import java.sql.SQLException;
12
13  public class Conexao extends Thread{
14
15      private DataInputStream in;
16      private DataOutputStream out;
17      private Socket clientSocket;
18      private Integer flag;
19      private String opcao_menu;
20
21      public Conexao(Socket aclientSocket){
22
23          try {
24              this.clientSocket = aclientSocket;
25              this.in = new DataInputStream(clientSocket.getInputStream()
    );
26              this.out =new DataOutputStream(clientSocket.getOutputStream
    ());
27              System.out.println("Conectado ao servidor: "+this.
    clientSocket.getLocalAddress());
28          } catch (IOException ex) {
29              Logger.getLogger(Conexao.class.getName()).log(Level.SEVERE,
     null, ex);}
30          }
31
32          @Override
33          public void run(){ // THREAD
34              if(!clientSocket.isConnected()){
35                  System.out.println("N o conectou");
36              }else{
```

```
37                  System.out.println("Endereco:"+clientSocket.
    getInetAddress().getHostAddress());
38              }
39              try{
40                  String jdbcUrl = "jdbc:sqlite:C:\\Users\\danie\\
    Documents\\Dist2\\BD\\db_OTF2020.db";
41                  Connection conn = DriverManager.getConnection(jdbcUrl);
42                  System.out.println("[STATUS]   Conectado ao servidor");
43                  out.writeUTF("[SERVIDOR]   Entre com email e senha");
44                  String email = in.readUTF();
45                  String senha = in.readUTF();
46                  ServidorControle SC = new ServidorControle(email,senha)
    ;
47                  SC.verificaAcesso(email,senha,conn);
48                  out.writeUTF("Login Aprovado");
49                  flag = 1;
50                  while(flag!=0){
51                      String confirmacao_login = in.readUTF();
52                      if(confirmacao_login.equals("continue")){
53                          SC.printaMenu();
54                          opcao_menu = in.readUTF();
55                          System.out.print("MENSAGEM DE CONFIRMA  O:"+
    opcao_menu);
56                          flag = 1;
57                      }
58                  }
59              }catch(SQLException e){
60                  System.out.println("Erro ao conectar ao banco de dados:
    "+e.getMessage());
61                  e.printStackTrace();
62              }catch(IOException e){
63                  System.err.println("Erro IOException"+e.getMessage());
64              }
65              try{
66              clientSocket.close();
67              System.out.println("Fechou uma conex o");
68              } catch (IOException ex) {
69              Logger.getLogger(Conexao.class.getName()).log(Level.SEVERE,
     null, ex);
70          }
71      }
72  }
```

3. Conclusão

Para a realização deste trabalho inicialmente precisamos fazer a conexão acontecer antes de começarmos outras partes do código, isso levou a um atraso do processo já que obtivemos uma maior dificuldade para fazer essa conexão, pois para versões diferentes do *Connector J* exigiam implementações diferentes, dificultando um pouco.

Outo ponto de dificuldade foi decidir entre MySql e o SQLite. Percebemos que o MySql tem uma maior complexidade para desenvolver em Java. E não conseguimos realizar a conexão dessa forma, mesmo com os materiais disponíveis na internet. Já o SQLite, que foi o escolhido, tem um caminho mais simples, com ele foi possível conectar

o banco de dados em uma apenas uma linha, o que deixou o programa mais leve e o código mais enxuto.

Portanto, com este trabalho conseguimos perceber o quão vasto são os caminhos para a construção de um sistema distribuído, além de entendermos também sua complexidade durante o processo de construção.

Iremos continuar a implementação da API de Sockets, visto que as intercorrências relatadas anteriormente, impossibilitaram de concluir esta parte no tempo hábil. Nas próximas partes do trabalho da disciplina, iremos apresentar a implementação da API de Sockets, mais as próximas implementações requisitadas.