ECM251 – Linguagens de Programação I

Aula 21 – L1/1, L2/1 e L3/1

Engenharia da Computação – 3ª série

Cliente-Servidor Java Sockets Bidirecional (L1/1, L2/1 e L3/1)

2024

Aula 21 – L1/1, L2/1 e L3/1

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Evandro*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro.*

Cliente-Servidor Java Sockets Bidirecional

Tópico

- Comunicação Bidirecional entre Cliente-Servidor Java

Definição

- Na Arquitetura Cliente-Servidor, *i.e. Client-Server Architecture*, é possível se estabelecer comunicação entre eles, com um **cliente** enviando dados para o **servidor** e, logo após, o **servidor** enviando dados de respostas para esse **cliente**;
- Por outro lado, na Arquitetura Cliente-Servidor, também é possível se estabelecer comunicação entre eles, com o **servidor** enviando dados para um **cliente** e, logo após, um **cliente** enviando dados de respostas ao **servidor**, estabelecendo o que é denominado de Comunicação **Bidirecional** entre **Servidor** e **Cliente** Java;

Comunicação Bidirecional entre Cliente-Servidor Java

Definição

- A comunicação **bidirecional** entre **cliente** e **servidor** em Java refere-se à capacidade de os **clientes** e **servidores** trocarem informações de maneira interativa, permitindo que ambos enviem e recebam dados ao longo da conexão de rede;
- Isso é particularmente útil em aplicativos em tempo real, como jogos *online*, aplicativos de bate-papo, *i.e. chat*, aplicativos de **streaming** de áudio/vídeo etc., onde as suas atualizações constantes, *i.e. status* e dados, precisam ser transmitidas entre o eles rapidamente.

Cliente-Servidor Java Sockets Bidirecional

Tópico

- Definições da Comunicação Bidirecional Cliente-Servidor Java

Definição

1. *Sockets*:
   - São a base da comunicação de rede em Java;
   - Permitem que os aplicativos estabeleçam conexões de rede e troquem dados entre si;
   - A classe *Socket* é usada pelos **clientes** para se conectar a um **servidor**, enquanto a classe *ServerSocket* é usada pelo **servidor** para esperar por conexões de **clientes**;

Definição

2. Protocolos:
   - Para que a comunicação seja eficaz, os **clientes** e **servidores** precisam seguir um **protocolo** de comunicação compartilhado;
   - Isso define como os dados são estruturados e como as mensagens são enviadas e recebidas;
   - Alguns protocolos comuns para comunicação bidirecional incluem o **TCP** – *Transmission Control Protocol* e o ***WebSocket***;

Definição

3. *Thread*:
   - Em muitos casos, é necessário usar ***threads*** para gerenciar a comunicação **bidirecional** de forma **eficiente**;
   - Isso ocorre porque a comunicação pode ser **bloqueante**, ou seja, o **servidor** precisa esperar por dados do **cliente**, e vice-versa;
   - Usar ***threads*** permite que o **servidor** atenda a vários **clientes** simultaneamente.

Definição

4. *Frameworks* e Bibliotecas:

   - A linguagem de programação **Java** oferece várias **bibliotecas** e ***frameworks*** para facilitar a implementação da **comunicação bidirecional**;
   - Alguns exemplos incluem: **Java *NIO***, ou ***New I/O***; **Apache *MINA***; ***Netty***; e **bibliotecas** específicas para ***WebSocket***.

Conclusões

- A implementação exata da comunicação **bidirecional** entre **cliente** e **servidor** em **Java** pode variar dependendo dos requisitos do aplicativo, do protocolo escolhido e das bibliotecas ou *frameworks* utilizados;
- No entanto, o conceito básico envolve a troca de dados interativa e em tempo real entre um **cliente** e um **servidor** por meio de uma **conexão** de rede.

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Servidor Java – bidirecional:

Classe *SimpleServerTest.java*

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Servidor Java – bidirecional:

```
1  import java.io.IOException;
2  import java.io.PrintStream;
3  import java.net.Socket;
4  import java.net.ServerSocket;
5  import java.util.Scanner;
6
7  public class SimpleServerTest
8  {  // Especifica endereco do IP local para o servidor
9     public  static  final String ENDERECO = "127.0.0.1";
10    // Especifica porta livre do computador para o servidor escutar
11    public  static  final int PORTA = 3334;
12
13    private static ServerSocket servidor;
14    private static Socket clienteAceito;
15    private static Scanner entrada;
16    private static PrintStream saida;
17
```

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Servidor Java – bidirecional (continuação):

```
18    public static void   main(String args[])
19    {  System.out.println("*v*v*v* CONSOLE DO SERVIDOR *v*v*v*");
20       try
21       {  // Inicia servidor e cria servico de escuta por porta via socket
22          iniciaServidor();
23          // Cria canal de comunicacao para conexao do cliente (servico)
24          aguardaConexaoCliente();
25          // Escutando e respondendo as mensagens do cliente
26          conversaComCliente();
27          // Fecha canal de comunicacao da conexao do cliente (servico)
28          encerraConexaoCliente();
29          // Fecha servico de escuta por porta via socket e encerra servidor
30          encerraServidor();
31       }
32       catch(IOException ex)
33       {  System.out.println("Erro no Servidor: " + ex.getMessage());
34       }
35    }
36
37    private static void iniciaServidor() throws IOException
38    {  // Cria servico de escuta pelo servidor na porta especificada via socket
39       servidor = new ServerSocket(PORTA);
40       System.out.println("Servidor iniciado e escutando a porta " + PORTA);
41    }
42
```

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Servidor Java – bidirecional (continuação):

```
43    private static void aguardaConexaoCliente() throws IOException
44    {  // Cria canal de comunicacao para conexao do cliente (servico)
45       clienteAceito = servidor.accept();
46       // Apresenta endereco do cliente conectado ao servidor
47       System.out.println("Cliente IP " +
48          clienteAceito.getInetAddress().getHostAddress() +
49          " conectado ao Servidor pela porta " + PORTA);
50       //Obtem a entrada do canal (socket)
51       entrada = new Scanner(clienteAceito.getInputStream());
52       // Objeto para enviar mensagem ao cliente
53       saida = new PrintStream(clienteAceito.getOutputStream());
54    }
55
56    private static void conversaComCliente() throws IOException
57    {  String msg;
58       // Escutando as mensagens do cliente que chegam ao servidor
59       while(entrada.hasNextLine())  // Aguarda proxima mensagem do cliente...
60       {  msg = leMensagemCliente();       // Le mensagem do Cliente
61          retornaMensagemCliente(msg);      // Servidor retorna mensagem ao cliente
62       }
63    }
64
```

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Servidor Java – bidirecional (continuação):

```
65    private static String leMensagemCliente() throws IOException
66    {  String msg = entrada.nextLine();  // Le mensagem do Cliente
67       System.out.print("Chegou do Cliente: ");
68       System.out.println(msg);   // Imprime na tela a mensagem de entrada
69       return msg;
70    }
71
72    private static void retornaMensagemCliente(String msg) throws IOException
73    {  // Servidor retorna mensagem ao Cliente
74       saida.println(msg);
75       System.out.print("Ecoou ao Cliente: ");
76       System.out.println(msg);   // Imprime na tela a mensagem de saÃ-da
77    }
78
79    private static void encerraConexaoCliente() throws IOException
80    {  // Fechando o canal de entrada do servidor e o servico de conexao ao servidor
81       entrada.close();  // Fecha o canal de comunicacao
82       System.out.println("Cliente se desconectou do Servidor!");
83    }
84
85    private static void encerraServidor() throws IOException
86    {  // Fecha servico de escuta por porta via socket e encerra servidor
87       servidor.close(); // Fecha o servico
88       System.out.println("Servidor finalizado!");
89    }
90 }
91
```

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Cliente Java – bidirecional:

Classe *SimpleClientTest.java*

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Cliente Java – bidirecional:

```
1  import java.io.IOException;
2  import java.io.PrintStream;
3  import java.net.Socket;
4  import javax.swing.JOptionPane;
5  import java.util.Scanner;
6
7  public class SimpleClientTest
8  {  // Criando a variavel de conexao do tipo Socket
9     private static Socket cliente;
10    private static Scanner entrada;
11    private static PrintStream saida;
12
13    public static void main(String args[])
14    {  System.out.println("*v*v*v* CONSOLE DO CLIENTE *v*v*v*");
15       try
16       {  // Cria Cliente e inicia conexao com Servidor via socket
17          iniciaCliente();
18          // Conversa com o Servidor atÃ© solicitar encerramento de comunicacao
19          conversaComServidor();
20          // Encerra conexao com o Servidor via socket
21          encerraConexaoServidor();
22       }
23       catch(IOException ex)
24       {  System.out.println("Erro no Cliente: " + ex.getMessage());
25       }
26    }
```

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Cliente Java – bidirecional (continuação):

```
27
28    private static void iniciaCliente() throws IOException
29    {  // Cria Cliente com IP e Porta para comunicacao com Servidor via socket
30       cliente = new Socket(SimpleServerTest.ENDERECO, SimpleServerTest.PORTA);
31       System.out.println("Cliente IP " + SimpleServerTest.ENDERECO +
32          " conectado ao Servidor pela porta " + SimpleServerTest.PORTA);
33       // Obtem a entrada do canal (socket)
34       entrada = new Scanner(cliente.getInputStream());
35    }
36
37    private static void conversaComServidor() throws IOException
38    {  String msg, echo = "";
39       // Conversa com o Servidor atÃ® solicitar encerramento de comunicacao
40       do
41       {  System.out.println("Digite na Entrada a mensagem para o Servidor!");
42          msg = JOptionPane.showInputDialog("Digite aqui a mensagem para o Servidor (ou <sair> para encerrar)");
43          if(!msg.equalsIgnoreCase("sair"))
44          {  enviaMensagemServidor(msg);
45             echo = leMensagemServidor();
46             verificaComunicacao(echo, msg);
47          }
48       }while(!msg.equalsIgnoreCase("sair"));
49    }
50
```

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Cliente Java – bidirecional (continuação):

```
51    private static void enviaMensagemServidor(String msg) throws IOException
52    {  // Objeto para enviar mensagem ao servidor
53       saida = new PrintStream(cliente.getOutputStream());
54       // Envia mensagem ao servidor
55       saida.println(msg);
56       System.out.print("Enviou ao Servidor: ");
57       System.out.println(msg);
58    }
59
60    private static String leMensagemServidor() throws IOException
61    {  String msg = entrada.nextLine();  // Le a mensagem recebida do Servidor
62       System.out.print("Ecoou do Servidor: ");
63       System.out.println(msg);   // Imprime na tela a mensagem de entrada
64       return msg;
65    }
66
67    private static void verificaComunicacao(String echo, String msg)
68    {  // Verifica se echo coincide com mensagem enviada...
69       if(echo.equals(msg))
70       {  // echo coincide, portanto, comunicacao OK!
71          System.out.println("Comunicacao OK!");
72       }
73       else
74       {  // echo nao coincide, portanto, comunicao NOK!
75          System.out.println("Comunicacao com problema!");
76       }
77    }
```

Cliente-Servidor Java Sockets Bidirecional

Exemplo

- Projeto do Cliente Java – bidirecional (continuação):

```
78
79    public static void encerraConexaoServidor()
80    {  // Encerra conexao com o Servidor via socket
81       System.out.println("Cliente se desconectou do Servidor!");
82       System.out.println("Cliente finalizado!");
83    }
84 }
85
```

Características

- Os códigos apresentados são aplicações simples da arquitetura **cliente-servidor** em **Java**;
- O código do **servidor** não apresenta interface gráfica, apenas mensagens de ***status***, via console;
- O código do **cliente** apresenta interface com o usuário simples, do tipo ***Swing***, além de apresentar mensagens de ***status***, via console, também;
- Essas aplicações não permitem conexões ao **servidor** por mais de um **cliente** por vez;
- Essas aplicações permitem o envio de mensagens de respostas do **servidor** para o **cliente**, quando iniciadas pelo **cliente**.

Conclusões

- O uso de aplicações **simples** em **Java** para arquitetura **cliente-servidor** pode ser uma escolha viável, dependendo dos requisitos do projeto e das necessidades específicas;
- A possibilidade de resposta ao **cliente**, enviada pelo **servidor**, abre um número grande de aplicações de comunicação entre **cliente-servidor**.

Cliente-Servidor Java Sockets Bidirecional

Exercício 1

a. Criar um projeto denominado ***SimpleClientServer*** na IDE de sua preferência e digitar as classes fornecidas ***SimpleServerTest.java*** e ***SimpleClientTest.java***;

b. Executar, somente, a classe ***SimpleClientTest.java***, verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre;

c. Executar a classe ***SimpleServerTest.java***, verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre;

d. Sem encerrar a execução da classe ***SimpleServerTest.java***, executar, a seguir, a classe ***SimpleClientTest.java***, digitando algumas mensagens na interface com o usuário, uma após a outra e, por fim, optar por sair. Verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre;

Cliente-Servidor Java Sockets Bidirecional

Exercício 1

e. Sem encerrar a execução da classe ***SimpleServerTest.java***, executar, a seguir, a classe ***SimpleClientTest.java***, digitando uma mensagem, apenas, na interface com o usuário e não optar por sair. Sem encerrar a execução da classe ***SimpleServerTest.java***, nem da primeira instância de execução da classe ***SimpleClientTest.java***, executar outra instância da classe ***SimpleClientTest.java***. Digitar algumas mensagens na segunda instância da classe ***SimpleClientTest.java***, sem sair dela, verificar e registrar o que ocorre. Repetir a digitação para a primeira instância da classe ***SimpleClientTest.java***, sem sair dela também. Explique, em detalhes, o que ocorre;

Exercício 1

f. Encerrar a execução da segunda instância da classe ***SimpleClientTest.java***, digitando algumas mensagens na primeira instância da classe ***SimpleClientTest.java***. Verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre;

g. Executar uma segunda instância da classe ***SimpleServerTest.java***, ainda com a primeira instância dessa classe em execução. Verificar e registrar o que ocorre. Explique, em detalhes, o que ocorre.

Exercício 2

- Criar uma interface gráfica para a classe ***SimpleClientTest.java***, através dos modelos de *layout* estudados anteriormente nesta disciplina (***flowlayout***, ***borderlayout***, ***gridlayout*** ou os automaticamente gerados pela IDE ***NetBeans***), eliminando toda e qualquer forma de comunicação com o usuário através do **console** e/ou por interface ***Swing*** como foi fornecida, criando nessa nova interface gráfica os campos específicos para as mensagens a serem enviadas ao servidor, respondidas ao cliente e para as mensagens de *status* da comunicação, além da inclusão dos botões e suas funcionalidades para **Enviar**, **Limpar** e **Sair**, na própria interface.

Exercício 3

- Baseada na solução do **Exercício 2** deste material, criar uma interface para o **servidor**, via **console**, onde será exibida a **mensagem** que deveria ser retransmitida automaticamente para seu **cliente** (vide o servidor do **exercício 2**), podendo, então, o usuário **editá-la**, ou **confirmá-la**, pelo próprio console do **servidor**, antes de ser reenviada para o **cliente**, criando para isso a classe ***SimpleServer2Test.java***.

Exercício 4

- Pesquisar, entender, estudar, resumir, exemplificar e dar aplicações, em PDF, o conceito de **Função *Lambda***, também conhecida por **Expressão *Lambda***, além de suas respectivas traduções literais para o idioma Inglês;
- Pesquisar, entender, estudar, resumir, exemplificar e dar aplicações, em PDF, o conceito de ***Threads* em Java**, também conhecida por ***Java Threads***.

Exercício Desafio

- Estudar, pesquisar, desenvolver e testar novas classes ***ServerTest.java***, ***ClientTest.java***, e as demais que se fizerem necessárias, baseadas nas respectivas classes fornecidas, fazendo com que a nova classe servidor permita a conexão de mais de um cliente ao mesmo tempo, gerenciando corretamente suas respectivas comunicações.

Bibliografia Básica

- MILETTO, Evandro M.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software II: introdução ao desenvolvimento web com HTML, CSS, javascript e PHP (Tekne). Porto Alegre: Bookman, 2014. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/9788582601969
- WINDER, Russel; GRAHAM, Roberts. Desenvolvendo Software em Java, 3ª edição. Rio de Janeiro: LTC, 2009. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/978-85-216-1994-9
- DEITEL, Paul; DEITEL, Harvey. Java: how to program early objects. Hoboken, N. J: Pearson, c2018. 1234 p. ISBN 9780134743356.

Continua...

Bibliografia Básica (*continuação*)

- HORSTMANN, Cay S; CORNELL, Gary. Core Java. SCHAFRANSKI, Carlos (Trad.), FURMANKIEWICZ, Edson (Trad.). 8. ed. São Paulo: Pearson, 2010. v. 1. 383 p. ISBN 9788576053576.
- LIANG, Y. Daniel. Introduction to Java: programming and data structures comprehensive version. 11. ed. New York: Pearson, c2015. 1210 p. ISBN 9780134670942.
- TURINI, Rodrigo. Desbravando Java e orientação a objetos: um guia para o iniciante da linguagem. São Paulo: Casa do Código, [2017]. 222 p. (Caelum).

Bibliografia Complementar

- HORSTMANN, Cay. Conceitos de Computação com Java. Porto Alegre: Bookman, 2009. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/9788577804078
- MACHADO, Rodrigo P.; FRANCO, Márcia H. I.; BERTAGNOLLI, Silvia de Castro. Desenvolvimento de software III: programação de sistemas web orientada a objetos em java (Tekne). Porto Alegre: Bookman, 2016. E-book. Referência Minha Biblioteca: https://integrada.minhabiblioteca.com.br/#/books/9788582603710
- BARRY, Paul. Use a cabeça! Python. Rio de Janeiro: Alta Books, 2012. 458 p.
  ISBN 9788576087434.

Continua...

Bibliografia Complementar (continuação)

- LECHETA, Ricardo R. Web Services RESTful: aprenda a criar Web Services RESTfulem Java na nuvem do Google. São Paulo: Novatec, 2015. 431 p. ISBN 9788575224540.
- SILVA, Maurício Samy. JQuery: a biblioteca do programador. 3. ed. rev. e ampl. São Paulo: Novatec, 2014. 544 p. ISBN 9788575223871.
- SUMMERFIELD, Mark. Programação em Python 3: uma introdução completa à linguagem Phython. Rio de Janeiro: Alta Books, 2012. 506 p. ISBN 9788576083849.

Continua...

Bibliografia Complementar (continuação)

- YING, Bai. Practical database programming with Java. New Jersey: John Wiley & Sons, c2011. 918 p.
- ZAKAS, Nicholas C. The principles of object-oriented JavaScript. San Francisco, CA: No Starch Press, c2014. 97 p. ISBN 9781593275402.
- CALVETTI, Robson. Programação Orientada a Objetos com Java. Material de aula, São Paulo, 2020.

Aula 21 – L1/1, L2/1 e L3/1

FIM

Engenharia da Computação – 3ª série

Cliente-Servidor Java Sockets Bidirecional (L1/2, L2/2 e L3/2)

2024

Aula 21 – L1/2, L2/2 e L3/2

Horário

Terça-feira: 2 x 2 aulas/semana

- L1/1 (07h40min-09h20min): *Prof. Calvetti*;
- L1/2 (09h30min-11h10min): *Prof. Calvetti*;
- L2/1 (07h40min-09h20min): *Prof. Evandro*;
- L2/2 (11h20min-13h00min): *Prof. Calvetti*;
- L3/1 (09h30min-11h10min): *Prof. Evandro*;
- L3/2 (11h20min-13h00min): *Prof. Evandro.*

Exercícios

- Terminar, entregar e apresentar ao professor para avaliação, os exercícios propostos na aula de teoria, deste material.

Bibliografia (apoio)

- LOPES, ANITA. GARCIA, GUTO. Introdução à Programação: 500 algoritmos resolvidos. Rio de Janeiro: Elsevier, 2002.
- DEITEL, P. DEITEL, H. Java: como programar. 8 Ed. São Paulo: Prentice-Hall (Pearson), 2010;
- BARNES, David J.; KÖLLING, Michael. Programação orientada a objetos com Java: uma introdução prática usando o BlueJ . 4. ed. São Paulo: Pearson Prentice Hall, 2009.

Aula 21 – L1/2, L2/2 e L3/2

FIM