Comunicação por Sockets

Natália Cardoso Gonçalves

Este trabalho tem como objetivo desenvolver uma aplicação em Java que pode se comunicar via rede local ou internet por sockets. Tal aplicação normalmente é composta por uma parte servidora e um ou mais clientes. Um cliente solicita determinado serviço ao servidor que processa a solicitação e devolve a informação ao cliente. Como muitos serviços podem ser disponibilizados numa mesma máquina, devem ser diferenciados por um endereço formato pelo endereço IP e o número da porta.

1. Protocolo

Da necessidade dos computadores se comunicarem surgiram diversos protocolos que permitem a troca de informação. Neste trabalho utilizou-se o TCP, o qual é importado diretamente pelo pacote java.net.

A vantagem de usar o TCP é que esse protocolo cuida para que os pacotes recebidos sejam remontados no host de destino na ordem correta (caso algum pacote não tenha sido recebido, o TCP requisita novamente este pacote). Somente após a montagem de todos os pacotes é que as informações ficam disponíveis para nossas aplicações. A programação do TCP com sockets utiliza streams, o que simplifica o processo de leitura e envio de dados pela rede.

Outra característica importante do TCP é que os pedidos de conexões dos clientes vão sendo mantidos em uma fila pelo sistema operacional até que o servidor possa atendê-los. Isto evita que o cliente receba uma negação ao seu pedido, pois o servidor pode estar ocupado com outro processo e não conseguir atender o cliente naquele momento.

Cada sistema operacional pode manter em espera um número limitado de conexões até que sejam atendidas. Quando o sistema operacional recebe mais conexões que esse limite, as conexões mais antigas vão sendo descartadas.

2. Troca de dados

A troca de dados entre clientes e servidor podem ser esquematicamente representada pela Figura 1.

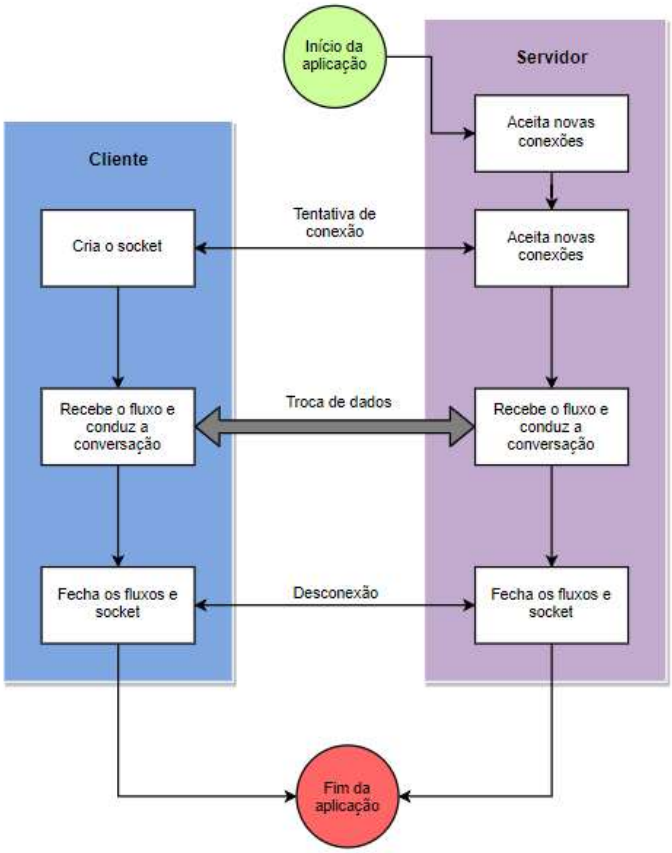

Figura 1 Troca de dados entre cliente e servidor

Inicialmente, o servidor é criado informando o número de porta do serviço oferecido.

```
public Server(int porta) {
    this.porta = porta;
    this.clientes = new ArrayList<PrintStream>();
}
```

Figura 2 Construtor do servidor

O número da porta pode variar entre 0 e 65535, porém, recomenda-se utilizar de 1024 em diante, pois as portas com números abaixo deste são reservados para o uso do sistema (por exemplo a porta 80 é usada pelo protocolo HTTP, 25 pelo SMTP, 110 pelo POP3, entre vários outros serviços).

Em seguida, deve-se estabelecer um socket para o servidor, pelo qual ele ficará "escutando" a porta destinada por solicitações de conexões. Esse socket recebe o nome de ServerSocket, no Java.

```
private void criarServerSocket() throws IOException {
    this.serverSocket = new ServerSocket(this.porta);
    System.out.println("Aguardando conexao na porta " + this.porta);
}
```

Figura 3 Criação do ServerSocket

Esta classe possui múltiplos construtores. Se considerarmos que existe apenas uma placa de rede, pode-se informar apenas o número da porta. Do contrário, é necessário realizar o *bind* com o IP da placa de rede que será utilizada. No código em questão, considerou-se apenas o primeiro caso.

```
private void esperaConexao() throws IOException {
    this.socket = serverSocket.accept();
    System.out.println("Connectando cliente..." + socket.getInetAddress().getHostAddress());

    PrintStream ps = new PrintStream(socket.getOutputStream());
    this.clientes.add(ps);

    TrataConexao tc = new TrataConexao(socket.getInputStream());
    new Thread(tc).start();
}
```

Figura 4 Espera por conexao

O método **accept()** impede o programa de continuar até que o cliente faça uma requisição. Ao aceita-la o cliente é adicionado a uma lista de dispositivos ativos e é criada uma thread para comunicação. No caso solicitado em que o servidor deverá estabelecer um diálogo com o cliente full-duplex, essa conexão requer apenas que seja um canal para inputs, como apresentado na figura 4. Contudo, caso o servidor fosse apenas um servidor de serviços e os clientes comunicassem entre si, também por full duplex, deve-se criar uma cópia completa do servidor, mantendo o canal original aberto para novas conexões, conforme apresentado na Figura 5.

```
public Server(Socket socket) {
    this.socket = socket;
}
```

```
Thread t = new Server(socket);
t.start();
```

Figura 5 Alterações no código para que o servidor seja passivo no chat

Quando um cliente é criado, deve informar o IP do servidor (host) e a porta com a qual se deseja conectar. Como o trabalho requer um chat, adicionou-se a informação de um nome/nickname pelo qual o cliente será identificado no diálogo.

```
public Client(String host, int porta, String nome) {
    this.host = host;
    this.porta = porta;
    this.nome = nome;
}
```

Figura 6 Construtor do cliente

Assim como o servidor, o cliente cria uma socket para comunicação e uma thread.

```
public void criaSocket() throws UnknownHostException, IOException {
    this.socket = new Socket(this.host, this.porta);
    System.out.println("Conexao estabelecida");
}

public void criaThread() throws IOException {
    RecebedorCliente r = new RecebedorCliente(this.socket.getInputStream());
    new Thread(r).start();
}
```

Estando estabelecida a comunicação, a troca de mensagens pode ocorrer livremente. Como pode-se enviar mais de uma mensagem por cliente ou servidor, optou-se por manter um Buffer para armazenamento temporário, o qual é consumido ao imprimir na tela a mensagem recebida. Caso o cliente queira sair da conversa, basta digitar o comando “/sair”. Este comando não desabilita o terminal que, poderá ser interrompido pelo mesmo comando.

```java
private void trataConexao() throws IOException{
    try {
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
        PrintStream output = new PrintStream(socket.getOutputStream());

        String linha = input.readLine();
        while (linha != null && !linha.contentEquals("/sair")){
            output.println(this.nome + ": " + linha);
            linha = input.readLine();
        }
        input.close();
        output.close();
    } finally {
        fechaSocket();
    }
}

private void fechaSocket() throws IOException {
    this.socket.close();
}
```

Figura 7 Troca de mensagens entre cliente e servidor

3. Código completo com o servidor ativo no chat

Neste caso, o servidor além de prover o serviço também atua como cliente, trocando mensagens com o cliente conectado. Embora não seja o comportamento usual de um servidor, foi um requisito do projeto.

```java
package wthread;

import java.io.InputStream;
import java.util.Scanner;

public class RecebedorCliente implements Runnable {

    private InputStream servidor;

    public RecebedorCliente(InputStream servidor) {
        this.servidor = servidor;
    }


    @Override
    public void run() {
        Scanner s = new Scanner(this.servidor);
        while(s.hasNextLine()) {
            System.out.println(s.nextLine());
        }
        s.close();
    }
}
```

```java
package wthread;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class Client {
    private Socket socket;
    private String host;
    private int porta;
    private String nome;

    public Client(String host, int porta, String nome) {
        this.host = host;
        this.porta = porta;
        this.nome = nome;
    }

    public void criaSocket() throws UnknownHostException, IOException {
        this.socket = new Socket(this.host, this.porta);
        System.out.println("Conexao estabelecida");
    }

    public void criaThread() throws IOException {
        RecebedorCliente r = new RecebedorCliente(this.socket.getInputStream());
        new Thread(r).start();
    }
```

```java
private void trataConexao() throws IOException{
    try {
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
        PrintStream output = new PrintStream(socket.getOutputStream());

        String linha = input.readLine();
        while (linha != null && !linha.contentEquals("/sair")){
            output.println(this.nome + ": " + linha);
            linha = input.readLine();
        }
        input.close();
        output.close();
    } finally {
        fechaSocket();
    }
}

private void fechaSocket() throws IOException {
    this.socket.close();
}

public String getNome() {
    return this.nome;
}

Run | Debug
public static void main(String[] args) throws UnknownHostException, IOException {
    System.out.println("Informe o host: ");
    String host = new Scanner(System.in).nextLine();

    System.out.println("Informe o nickname: ");
    String nickname = new Scanner(System.in).nextLine();

    Client cliente = new Client(host, 2525, nickname);
    cliente.criaSocket();
    cliente.criaThread();
    cliente.trataConexao();
}
}
```

```java
package wthread;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class Server {
    private ServerSocket serverSocket;
    private int porta;
    private List<PrintStream> clientes;
    private Socket socket;

    public Server(int porta) {
        this.porta = porta;
        this.clientes = new ArrayList<PrintStream>();
    }


    private void criarServerSocket() throws IOException {
        this.serverSocket = new ServerSocket(this.porta);
        System.out.println("Aguardando conexao na porta " + this.porta);
    }


    private void esperaConexao() throws IOException {
        this.socket = serverSocket.accept();
        System.out.println("Connectando cliente..." + socket.getInetAddress().getHostAddress());

        PrintStream ps = new PrintStream(socket.getOutputStream());
        this.clientes.add(ps);

        TrataConexao tc = new TrataConexao(socket.getInputStream());
        new Thread(tc).start();
    }
```

```java
    private void trataConexao() throws IOException{
        try {
            BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
            PrintStream output = new PrintStream(socket.getOutputStream());

            String linha = input.readLine();
            while (linha != null && !linha.contentEquals("/sair")){
                output.println("Server: " + linha);
                linha = input.readLine();
            }

            input.close();
            output.close();
        } finally {
            fechaSocket();
        }
    }

    private void fechaSocket() throws IOException {
        this.socket.close();
    }

    Run | Debug
    public static void main(String[] args) throws IOException {
        Server servidor = new Server(2525);
        servidor.criarServerSocket();
        servidor.esperaConexao();
        servidor.trataConexao();
    }
}
```

4. Código completo com o servidor passivo no chat e múltiplos clientes

Neste caso, o servidor apenas faz broadcast de todas as mensagens recebidas dos clientes. Os clientes trocam mensagens entre si e a entrada e/ou saída de cada cliente não interfere no serviço.

```java
package server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Enumeration;
import java.util.Vector;

public class Server extends Thread {

    private static Vector<PrintStream> clientes;
    private Socket socket;
    private String name;

    public Server(Socket socket) {
        this.socket = socket;
    }


    public void broadCast(PrintStream output, String verbo, String linha) throws IOException {
        Enumeration e = clientes.elements();

        while (e.hasMoreElements()) {

            PrintStream chat = (PrintStream) e.nextElement();

            if (chat != output) {
                chat.println(name + verbo + linha);
            }
        }
    }
```

```java
@Override
public void run() {

    try {

        BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));
        PrintStream output = new PrintStream(socket.getOutputStream());

        name = input.readLine();

        if (name == null) {
            return;
        }

        clientes.add(output);

        String linha = input.readLine();

        while (linha != null && !(linha.trim().equals(""))) {
            broadCast(output, " diz: ", linha);
            linha = input.readLine();
        }

        broadCast(output, " saiu: ", " do chat!");
        clientes.remove(output);
        socket.close();

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

}
```

```
    Run | Debug
    public static void main(String[] args) {


        clientes = new Vector();

        try {

            ServerSocket server = new ServerSocket(2000);


            while (true) {

                System.out.println("Esperando conexoes");
                Socket socket = server.accept();
                System.out.println("Nova conexao");


                Thread t = new Server(socket);
                t.start();

            }

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

    }
}
```

```java
package server;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;


public class Client extends Thread {

    private static boolean connection = false;
    private Socket socket;

    public Client(Socket socket) {
        this.socket = socket;
    }

    @Override
    public void run() {
        try {
            BufferedReader input = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            while (true) {
                String linha = input.readLine();

                if (linha == null) {
                    System.out.println("Fim da conexao");
                    break;
                }

                System.out.println(linha);
                System.out.println("> ");

            }

        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
        connection = true;
    }
```

```java
    Run | Debug
    public static void main(String[] args) {
        try {


            Socket socket = new Socket("127.0.0.1", 2000);


            BufferedReader keyboad = new BufferedReader(new InputStreamReader(System.in));
            PrintStream output = new PrintStream(socket.getOutputStream());

            System.out.println("Insira seu nome: ");
            String name = keyboad.readLine();
            output.println(name);


            Thread t = new Client(socket);
            t.start();


            while (true) {
                System.out.print("> ");
                String linha = keyboad.readLine();


                if (connection) {
                    break;
                }
                output.println(linha);
            }

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

}
```