

UNIVERSIDADE FEDERAL DO CEARÁ
REDES DE COMPUTADORES
SISTEMAS DISTRIBUÍDOS
FRANCISCO JARBAS DOS SANTOS SOUSA

RELATÓRIO: LABORATÓRIO JAVA RPC SEM RMI

QUIXADÁ-CE

2025

Sumário

INTRODUÇÃO.....	3
DESENVOLVIMENTO DO CÓDIGO.....	3
INTERFACE RPC.....	3
IMPLEMENTAÇÃO DO SERVIDOR.....	4
IMPLEMENTAÇÃO DO CLIENTE.....	5
CONCLUSÃO.....	5

INTRODUÇÃO

Este laboratório demonstra a implementação de um sistema de Remote Procedure Call (RPC) utilizando Java e sockets, sem o uso do Java RMI (Remote Method Invocation). A abordagem adotada simula chamadas remotas de métodos através de comunicação por sockets, permitindo a interação entre um cliente e um servidor.

CONFIGURAÇÃO DO PROJETO

O projeto é estruturado como um projeto Maven, com um arquivo pom.xml básico contendo as configurações essenciais, como o Group ID, Artifact ID e versão.

Exemplo de pom.xml:

```
1 http://maven.apache.org/xsd/maven4.0.0.xsd (xsi:schemaLocation)
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>com.exemplo</groupId>
7   <artifactId>JavaRPC-Sockets</artifactId>
8   <version>1.0-SNAPSHOT</version>
9 </project>
```

DESENVOLVIMENTO DO CÓDIGO

A implementação segue a seguinte estrutura:

- Interface RPC (HelloService.java)
- Implementação do Serviço (HelloServiceImpl.java)
- Servidor (Server.java)
- Cliente (Client.java)

INTERFACE RPC

A interface HelloService define o contrato do serviço RPC. O método sayHello(String name) é declarado para ser chamado remotamente.

```

1 package com.exemplo;
2
3 // Interface do serviço
4 public interface HelloService {
5     String sayHello(String name);
6 }

```

IMPLEMENTAÇÃO DO SERVIÇO

A classe HelloServiceImpl implementa a interface HelloService, fornecendo uma lógica simples para responder às requisições.

```

1 package com.exemplo;
2
3 // Implementação do serviço
4 public class HelloServiceImpl implements HelloService {
5     @Override
6     public String sayHello(String name) {
7         return "Olá, " + name + "! Este é um exemplo de RPC com sockets.";
8     }
9 }

```

IMPLEMENTAÇÃO DO SERVIDOR

O servidor escuta conexões na porta 5000, processa requisições e retorna respostas conforme a chamada remota.

```

1 package com.exemplo;
2
3 import java.io.*;
4
5 public class Server {
6     public static void main(String[] args) {
7         try (ServerSocket serverSocket = new ServerSocket(5000)) {
8             System.out.println("Servidor pronto na porta 5000...");
9
10            // Espera por conexões
11            while (true) {
12                Socket clientSocket = serverSocket.accept();
13                System.out.println("Cliente conectado!");
14                try (BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
15                    PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) {
16
17                    // Recebe a requisição
18                    String input = in.readLine();
19                    System.out.println("Requisição recebida: " + input);
20                    // Processa a chamada ao método
21                    if (input.startsWith("sayHello:")) {
22                        String name = input.split(":")[1];
23                        HelloService service = new HelloServiceImpl();
24                        String response = service.sayHello(name);
25                        // Envia a resposta de volta ao cliente
26                        out.println(response);
27                    } else {
28                        out.println("Método não suportado.");
29                    }
30                }
31                clientSocket.close();
32            }
33        } catch (IOException e) {
34            e.printStackTrace();
35        }
36    }
37 }

```

IMPLEMENTAÇÃO DO CLIENTE

O cliente estabelece uma conexão com o servidor e faz a chamada RPC enviando uma string formatada.

```
1 package com.exemplo;
2
3 import java.io.*;
4
5
6 public class Client {
7     public static void main(String[] args) {
8         try (Socket socket = new Socket("localhost", 5000);
9             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
10             PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {
11
12             // Envia a chamada do método para o servidor
13             String request = "sayHello:Usuário";
14             out.println(request);
15             System.out.println("Requisição enviada: " + request);
16             // Recebe a resposta do servidor
17             String response = in.readLine();
18             System.out.println("Resposta do servidor: " + response);
19         } catch (IOException e) {
20             e.printStackTrace();
21         }
22     }
23 }
```

CONCLUSÃO

O laboratório demonstra como é possível implementar um sistema de chamadas remotas usando sockets em Java, sem depender de RMI. O modelo adotado é simples, mas eficaz para entender conceitos básicos de RPC. No entanto, em sistemas reais, protocolos mais robustos como gRPC ou RESTful APIs são geralmente preferidos para maior escalabilidade e segurança.