Sistemas Distribuídos
PPGCC – PUCRS – Abril/2007

Exemplos práticos do uso de RMI em sistemas distribuídos

Elder de Macedo Rodrigues
Guilherme Montez Guindani
Leonardo Albernaz Amaral
Fábio Delamare

Cronograma

- **Introdução**
- **Aplicação calculadora**
  - **RMI**
  - **Sockets**
- **Aplicação chat**
  - **Sockets**
  - **RMI**
- **Conclusão**

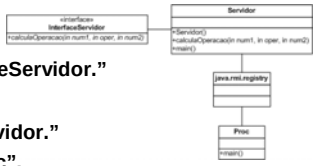
Introdução

- **Comunicação entre processos nos sistemas distribuídos (mais usados):**
  - **Sockets**
  - **RMI**
- **Objetivo do trabalho:**
  - **Abordar dois exemplos práticos (Chat e Calculadora)**
  - **Comparar tais exemplos em implementações diferentes (sockets e RMI)**

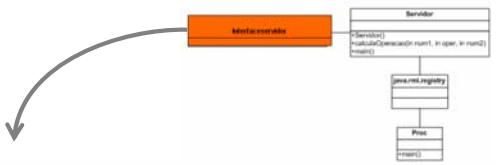
Calculadora RMI

Implementação

- **Interface:**
  - **Interface InterfaceServidor.”**
- **Classes:**
  - **Uma classe “Servidor.”**
  - **Uma classe “Proc”.**



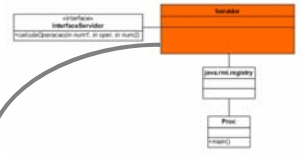
Implementação Interface




```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface InterfaceServidor extends Remote{

    public double calculaOperacao(int num1, char oper, int num2) throws RemoteException;

}
```

Implementação do Servidor

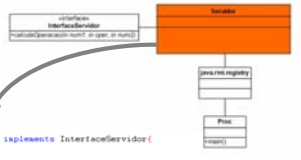
1. **Implementa o Servidor.**
2. Interface para método remoto.



```
/** Método principal do Servidor. Chama o método construtor da classe */
public static void main(String args[]) throws RemoteException{

    try {
        //Definir o security manager:
        if(System.getSecurityManager() == null){
            System.setSecurityManager(new RMISecurityManager());
        }

        System.out.println("Construindo as implementacoes do Servidor...");
        Servidor servidor = new Servidor();
        System.out.println("Registrando as implementacoes...");
        Naming.rebind("servidor", servidor);
        System.out.println("Servidor iniciado!");

        //System.exit(0);
```

Implementação do Servidor

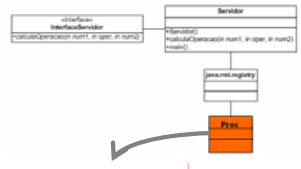
1. Implementa o Servidor.
2. **Interface para método remoto.**



```
public class Servidor extends UnicastRemoteObject implements InterfaceServidor{

    /** Método construtor */
    public Servidor() throws RemoteException{
        super();
    }

    /** Interface implementada para método remoto */
    public double calculaOperacao(int num1, char oper, int num2) throws RemoteException{

        double resultado = 0.0;
        switch(oper){
            case '+':
                resultado = num1 + num2;
                break;
            case '-':
                resultado = num1 - num2;
                break;
            case '*':
                resultado = num1 * num2;
                break;
            case '/':
                resultado = num1 / num2;
                break;
        }
        return resultado;
```

Implementação do Cliente

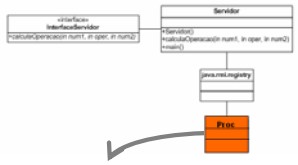
1. **Cria referência para o servidor remoto.**
2. Cria objeto de leitura do console.
3. Cria a interface com console e lê dados do console.
4. Envia dados lidos do console para o servidor através do canal de saída e recebe o resultado do servidor.



```
System.out.println("Registrando-se no servidor...");
Remote referenciaRemota = Naming.lookup("servidor");
InterfaceServidor a = (InterfaceServidor) referenciaRemota;
```

Implementação do Cliente

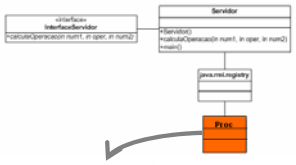
1. Cria referência para o servidor remoto.
2. **Cria objeto de leitura do console.**
3. **Cria a interface com console e lê dados do console.**
4. Envia dados lidos do console para o servidor através do canal de saída e recebe o resultado do servidor.



```
BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Sequencia esperada: Num1 (inteiro) + operador (+, -, *, /) + Num2 (inte
System.out.print("\nNum1 > ");
num1 = Integer.parseInt(teclado.readLine());
System.out.print("Oper > ");
oper = teclado.readLine().charAt(0);
System.out.print("Num2 > ");
num2 = Integer.parseInt(teclado.readLine());
```

Implementação do Cliente

1. Cria referência para o servidor remoto.
2. Cria objeto de leitura do console.
3. Cria a interface com console e lê dados do console.
4. **Envia dados lidos do console para o servidor através do canal de saída e recebe o resultado do servidor.**



```
resultado = a.calculaOperacao(num1, oper, num2);
System.out.println("-----------");
System.out.println("resultado > " + resultado);
```

```
Registrando-se no servidor...
Sequencia esperada: Num1 (inteiro) + operador (+, -, *, /) + Num2 (inteiro)

Num1 > 1
Oper > +
Num2 > 3_
```

Calculadora Sockets

Implementação Server

1. **Cria o ServerSocket;**
2. Cria objetos que controlam o fluxo de comunicação;
3. Recebe valores a serem calculados;
4. Calcula o resultado da operação desejada pelo cliente;
5. Envia para o cliente o resultado da operação;
6. Encerra os canais de comunicação e o socket com o cliente;

```
C:\j2sdk1.4.2_14\bin>javac CalcServer.java

C:\j2sdk1.4.2_14\bin>java CalcServer
Esperando conexao do cliente...
Esperando conexao do cliente...
```

```
//Cria o ServerSockets;
ss = new ServerSocket(5000);
while(true){
    System.out.println("Esperando conexao do cliente...");
    //Cria o Socket e espera por conexões;
    Socket s = ss.accept();
    //Cria Thread para socket com um cliente. Passa o socket
    Thread t = new CalcServer(s);
    t.start();
    //Volta ao loop para esperar mais concexões;
```

Implementação Server

1. Cria o ServerSocket;
2. **Cria objetos que controlam o fluxo de comunicação;**
3. **Recebe valores a serem calculados;**
4. **Calcula o resultado da operação desejada pelo cliente;**
5. **Envia para o cliente o resultado da operação;**
6. **Encerra os canais de comunicação e o socket com o cliente;**

```
2  //Cria objetos que controlam o fluxo de comunicação;
   DataInputStream in = new DataInputStream(conexao.getInputStream());
   DataOutputStream out = new DataOutputStream(conexao.getOutputStream());

3  //Recebe valores a serem calculados;
   num1 = in.readInt();
   oper = in.readChar();
   num2 = in.readInt();

4  //Calcula o resultado da operação desejada pelo cliente;
   resultado = calculaValor(num1, oper, num2);

5  //Envia para o cliente o resultado da operação;
   out.writeDouble(resultado);

6  //Encerra os canais de comunicação e o socket com o cliente;
   in.close();
   out.close();
   conexao.close();
```

Implementação Cliente

1. **Solicita um socket ao servidor.**
2. **Cria os canais de comunicação**
3. **Cria objeto de leitura do console**
4. **Cria interface com console**
5. **Envia dados lidos do console**
6. **Recebe do servidor o resultado**

```
//Solicita um socket ao servidor. Deve existir um socket esperando no lado do servidor.
Socket conexao = new Socket("localHost", 5000);

//Cria os canais de comunicação com o servidor:
DataInputStream in = new DataInputStream(conexao.getInputStream());
DataOutputStream out = new DataOutputStream(conexao.getOutputStream());

//Cria objeto de leitura do console (teclado):
BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));

//Cria interface com console (teclado) e lê dados do console (teclado);
System.out.println("Sequencia esperada: Num1 (inteiro) + operador (+, -, *, /) + Num2 (inteiro)");
System.out.print("\nNum1 ) ");
num1 = Integer.parseInt(teclado.readLine());
System.out.print("Oper ) ");
oper = teclado.readLine().charAt(0);
System.out.print("Num2 ) ");
num2 = Integer.parseInt(teclado.readLine());

//Envia dados lidos do console (teclado) para o servidor através do canal de saída (out).
out.writeInt(num1);
out.writeChar(oper);
out.writeInt(num2);

//Recebe do servidor o resultado da operação solicita.
resultado = in.readDouble();
System.out.println("=========");
System.out.println("resultado ) " + resultado);
```

```
C:\j2sdk1.4.2_14\bin>javac CalcClient.java

C:\j2sdk1.4.2_14\bin>java CalcClient
Sequencia esperada: Num1 (inteiro) + operador (+, -, *, /) + Num2 (inteiro)

Num1 ) 1
Oper ) +
Num2 ) 2
=========
resultado ) 3.0
```

Aplicação Chat

Implementação Chat - Sockets

- **Características do Servidor:**
  - **Multithread**
    - **Uma thread para cliente conectado**
  - **Armazena os sockets dos clientes em um vetor**
  - **Disponibiliza o serviço na porta 5000**
  - **Recebe mensagens de um cliente e as repassa em "broadcast" para os demais**

Implementação Chat - Sockets

- **Características do Cliente:**
  - **Multithread**
    - **Uma thread para recebimento de mensagens**
    - **Uma thread para envio de mensagens**
  - **Acessa o serviço na porta 5000**
  - **Lê mensagens do teclado e as envia para o servidor**
  - **Recebe mensagens do servidor e as imprime na tela**

Implementação Chat - Sockets

- **Execução do servidor**

```
D:\Temp\programação\java\sockets1\src\chat>java ChatServer
Esperando alguUm se conectar...
Algun cliente se concectou...
Esperando alguUm se conectar...
Algun cliente se concectou...
Esperando alguUm se conectar...
```

```
ServerSocket ss = new ServerSocket(5000);
while(true){
    System.out.println("Esperando alguem se conectar...");
    Socket s = ss.accept();
    System.out.println("Cliente conectado...");

    //Cria uma nova Thread para tratar essa conexão:
    Thread t = new ChatServer(s);
    t.start();
    //Volta ao loop para esperar mais conexões
}
```

```
//Coloca-se o canal de saida para o cliente no vetor:
clientes.add(out);

//Lê a mensagem recebida no canal de entrada:
String linha = in.readLine();

//Verifica-se se linha recebida não é null:
while(linha != null && !(linha.trim().equals(""))){
    //reenvia a linha para todos os clientes conectados:
    sendToAll(out, " disse: ", linha);
    //espera por uma nova linha:
    linha = in.readLine();
}
```

Implementação Chat - Sockets

- **Execução do Cliente**

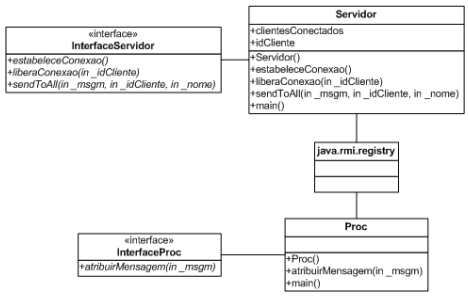
```
//enviar antes de tudo o nome do usuário:
BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));
System.out.print("Entre com o seu nome: ");
String NomeCliente = teclado.readLine();
saida.println(NomeCliente);

//uma vez que tudo está pronto, antes de iniciar o loop principal,
// executar a thread de recepção de mensagens:
Thread t = new ChatCliente(conexao);
t.start();

//loop principal: obtendo uma linha digitada no teclado e enviando-a
//para o servidor;
String linha;

while(true){
    // ler a linha digitada no teclado
    System.out.print("> ");
    linha = teclado.readLine();
    // antes de enviar, verifica se a conexão não foi fechada
    if(done){
        break;
    }
    // envia para o servidor
    saida.println(linha);
}
```

```
D:\Temp\programação\java\sockets
Entre com o seu nome: leonardo
> oi
>
Guilherme disse: ola
```

```
D:\Temp\programação\java\sockets
Entre com o seu nome: Guilherme
>
leonardo disse: oi
ola
```

Implementação Chat - RMI

- **Classes criadas:**

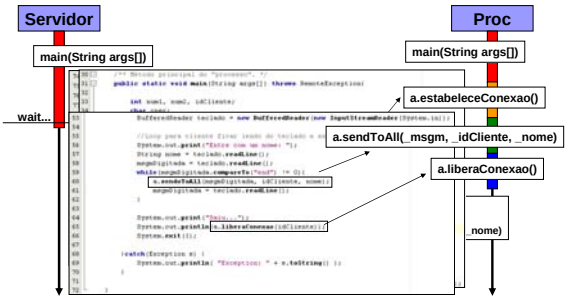


Implementação Chat - RMI

- **Características do Servidor (Servidor):**
  - **Implementa 3 métodos de interface**
  - **Disponibiliza esses métodos**
- **Características do Cliente (Proc):**
  - **Implementa 1 método de interface**
  - **Disponibiliza esse método**
  - **Interage com o usuario (lê e imprime mensagens)**

Implementação Chat - Sockets

- **Execução do chat em RMI**



Conclusões:

- **Número de linhas código**
- **Abstração da camada de comunicação**
- **Facilidade de uso e funcionalidades**

Dúvidas?