SISTEMAS DISTRIBUÍDOS
AULA 10

Prof.º Flávio Grassi
flavio.grassi@uni9.pro.br

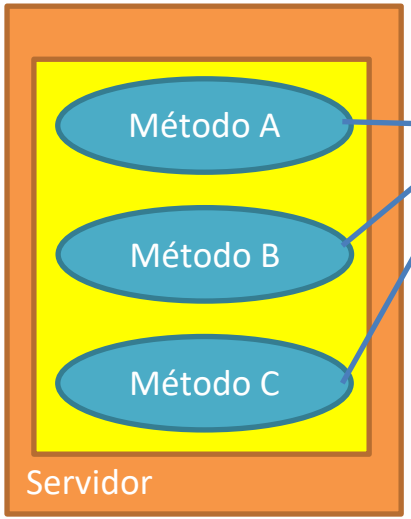
UNINOVE
Universidade Nove de Julho
A UNINOVE COM VOCÊ É 10

COMUNICAÇÃO EM REDE: RPC

Comunicação em Rede

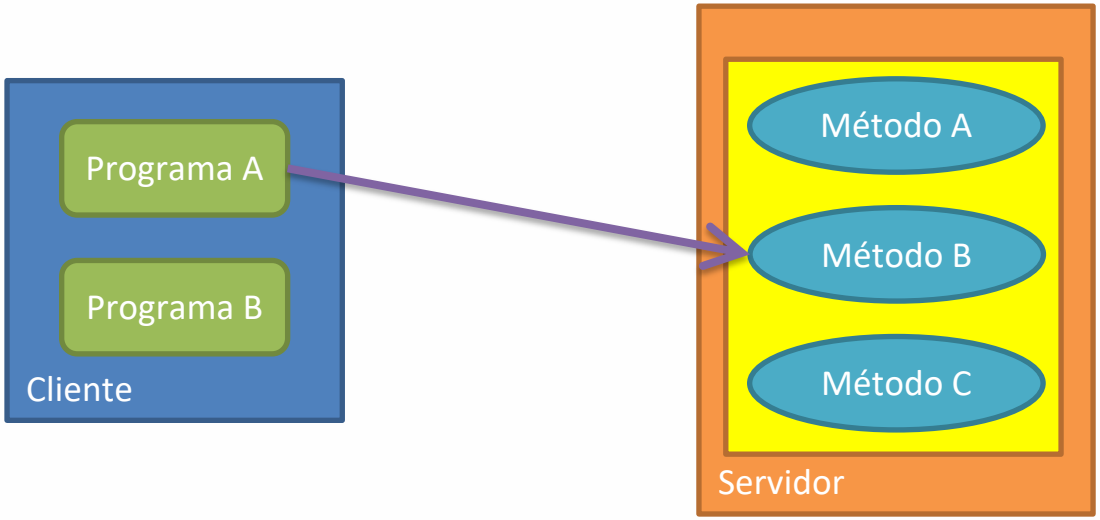
- *Remote Procedure Call* (RPC)
  - Nome genérico para representar uma forma de comunicação entre processos remotos (entre máquinas em rede).

RPC

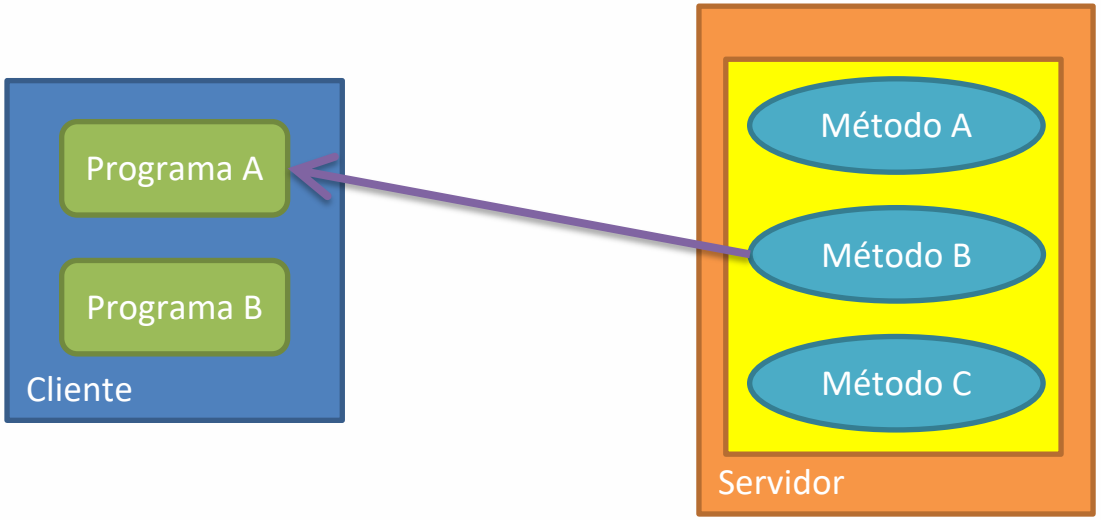


Partes
de um programa

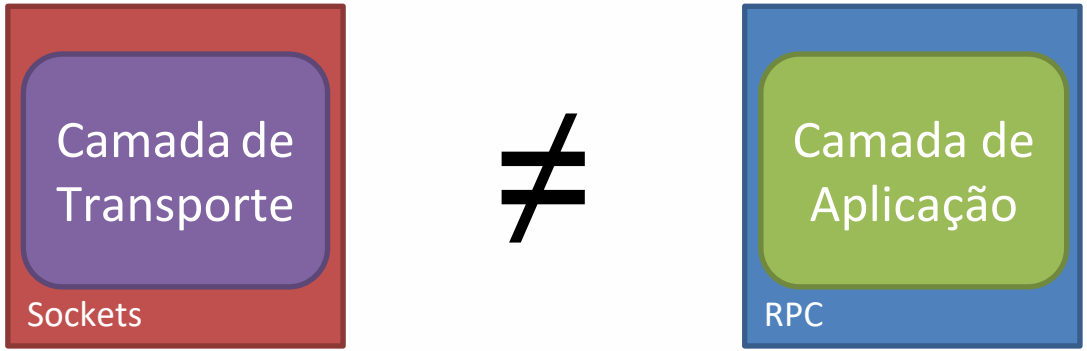
RPC



RPC



RPC



RPC

- A chamada desses métodos acontece como se os mesmos existissem localmente (na mesma máquina “cliente”).

RPC

- **Alta** granularidade.

RPC

- Não necessita, necessariamente, que os programas e métodos estejam em uma mesma linguagem de programação.

RPC

- Tipos de “RPC”

RPC

- Tipos de “RPC”
  - RPC *(Remote Procedure Call)*
    - C/C++, Pascal, Cobol, etc.

RPC

- Tipos de “RPC”
  - RPC *(Remote Procedure Call)*
    - C/C++, Pascal, Cobol, etc.
  - RMI (*Remote Method Invocation*)
    - Java, C#, Python, etc.

RPC

- Tipos de “RPC”
  - RPC *(Remote Procedure Call)*
    - C/C++, Pascal, Cobol, etc.
  - RMI (*Remote Method Invocation*)
    - Java, C#, Python, etc.
  - MOM (*Message Oriented Middleware*)
    - JSON/XML, etc.

Implementações de “RPC”

- Exemplos de frameworks:
  - RPyC
  - .NET Remoting
  - Java RMI

IMPLEMENTANDO VIA JRMI

JRMI

1. Criar um **interface** que será implementada, tanto nas máquinas cliente, quanto no servidor.

InterfaceRMI.java

```java
package jrmi;

import java.rmi.Remote;

public interface InterfaceRMI extends Remote {

    public String cumprimentar(String nome) throws Exception;

}
```

JRMI

2. Criar um classe **servidora**, que implementa a interface anterior e ”amarra” o endereço do servidor com um **nome de um objeto** (que será compartilhado via rede).

Servidor.java | Cliente.java

```java
package jrmi;

import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;

public class Servidor extends UnicastRemoteObject implements InterfaceRMI {

    public static void main(String[] args) {
        try {
            System.setProperty("java.rmi.server.hostname", "10.117.14.113");
            LocateRegistry.createRegistry(1099);
            Naming.rebind("//10.117.14.113/objetoCompartilhado", new Servidor());
            System.out.println("O servidor está pronto!");
        } catch (Exception e) {
            System.err.println("Falha ao iniciar o servidor!");
        }
    }

    public Servidor() throws Exception {
    }

    @Override
    public String cumprimentar(String nome) throws Exception {
        System.out.println(nome + " está entrando em contato!");
        return "Olá " + nome + ". Aqui é o servidor!";
    }
}
```

JRMI

3. Criar um classe **cliente**, que procura por uma **referência do objeto remoto**.
   - Uma vez encontrada essa referência, a comunicação entre o(s) cliente(s) e o servidor pode ser estabelecida, como se uma instância do servidor existisse na máquina cliente.

Cliente.java

```java
package jrmi;

import java.rmi.Naming;
import java.util.Scanner;

public class Cliente {

    public static void main(String[] args) throws Exception {
        InterfaceRMI objetoRemoto = (InterfaceRMI) Naming.lookup("//10.117.14.113/objetoCompartilhado");
        Scanner teclado = new Scanner(System.in);
        System.out.print("Digite seu nome: ");
        String nome = teclado.nextLine();
        System.out.println(objetoRemoto.cumprimentar(nome));
    }
}
```

Rodando em máquinas diferentes

- Cada máquina precisa ter – **localmente** – o seu código compilado **E** a interface.
  - Ou seja, a Classe “Cliente” (ou Classe “Servidora”) + “Interface”.

DEMONSTRAÇÃO EM SALA

Atividade sugerida

- Escrevam e testem/executem o código do slide, pois teremos atividade (em grupo) de 0 à 10 na próxima aula, onde vocês deverão implementar alterações no código, **sem ajuda do professor**.

OBRIGADO.

UNINOVE
Universidade Nove de Julho
A UNINOVE COM VOCÊ É 10