Sistemas Distribuídos

Anhanguera

AVALIE SUA PROFISSÃO

QUANDO APARECER EM SEU PORTAL UMA AVALIAÇÃO SOBRE SEU CURSO, RESPONDA:

NOTAS **9 OU 10** SIGNIFICA QUE VOCÊ INDICA

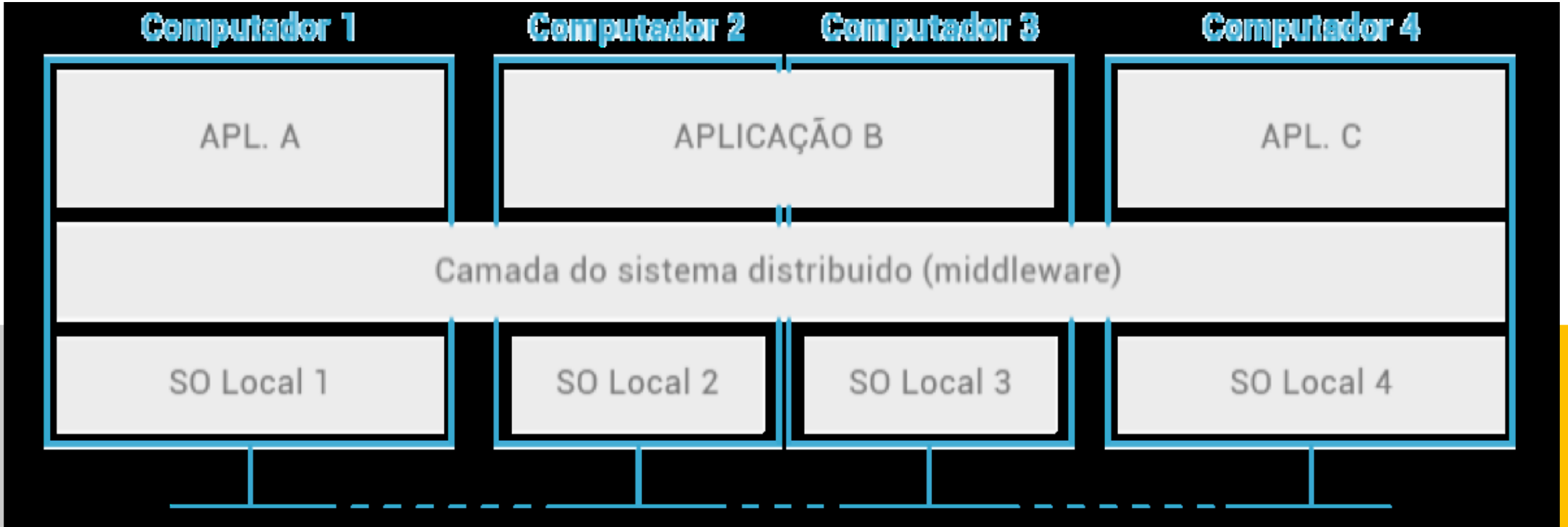
NOTAS **7 OU 8** SIGNIFICA QUE VOCÊ NÃO INDICA

Anhanguera

Anhanguera

Utilizando RPC com Java

Na comunicação entre máquinas em um sistema distribuído, é comum o uso de middlewares que, conforme foi estudado, servem como uma camada de abstração entre a chamada de métodos de alto nível – pelas aplicações – e a execução de métodos de baixo nível, dependentes do sistema operacional especificamente instalado naquela máquina. A Figura 4.12 representa essa camada de middleware no contexto de sistemas distribuídos.



Podemos observar na Figura 4.12 a camada de middleware que está interligando aplicações e sistemas operacionais que estão sendo executados em diferentes computadores.

Esse middleware nada mais é o do que a implementação – através de algum framework – de um modelo de comunicação entre máquinas conhecido genericamente por RPC, do inglês Remote Procedure Call. Em outras palavras, o RPC é uma forma de comunicação entre máquinas mais granular que a comunicação via sockets (COULOURIS, 2013). Como visto na Seção anterior, a comunicação via sockets envolve as camadas um a quatro do modelo de referência ISO/OSI. Já a comunicação via RPC envolve as sete camadas do modelo de referência ISO/OSI, e é uma forma mais granular de comunicação entre máquinas, uma vez que, diferentemente da comunicação via sockets, na qual executamos toda a aplicação, através do uso de RPC, podemos executar apenas um (ou mais) métodos de interesse, implementados em uma máquina, através de outra máquina.

Há de se salientar que, apesar de, na prática, utilizarmos esse termo quando queremos nos referir a esse tipo de comunicação entre máquinas, o termo RPC em si é, tecnicamente, apenas uma das formas de comunicação entre máquinas com alta granularidade, conforme observa Coulouris et al. (2013), sendo, inclusive, a mais antiga delas (porém ainda utilizada). Ainda conforme Coulouris, existem três modelos de comunicação entre máquinas, conforme o Quadro 4.2.

Modelo de Comunicação	Implementado através de
RPC *(Remote Procedure Call)*	Linguagens de programação estruturadas.
RMI *(Remote Method Invocation)*	Linguagens de programação orientadas a objeto.
MOM *(Message Oriented Midleware)*	Linguagens de programação para web.

Dependendo da linguagem de programação a qual o desenvolvedor tem mais familiaridade, existem vários frameworks que podem ser adotados para implementação do RPC. Abaixo são listados alguns dos mais utilizados, para algumas das linguagens de programação mais populares atualmente:

- Java:

JRMI <https://docs.oracle.com/javase/tutorial/rmi/overview.html>
JMS <https://www.oracle.com/technetwork/articles/java/introjms-1577110.html>

- C#:

.NET Remoting <https://msdn.microsoft.com/en-us/library/ms973857.aspx>
Akka.net https://getakka.net/articles/intro/tutorial-1.html

- Python:

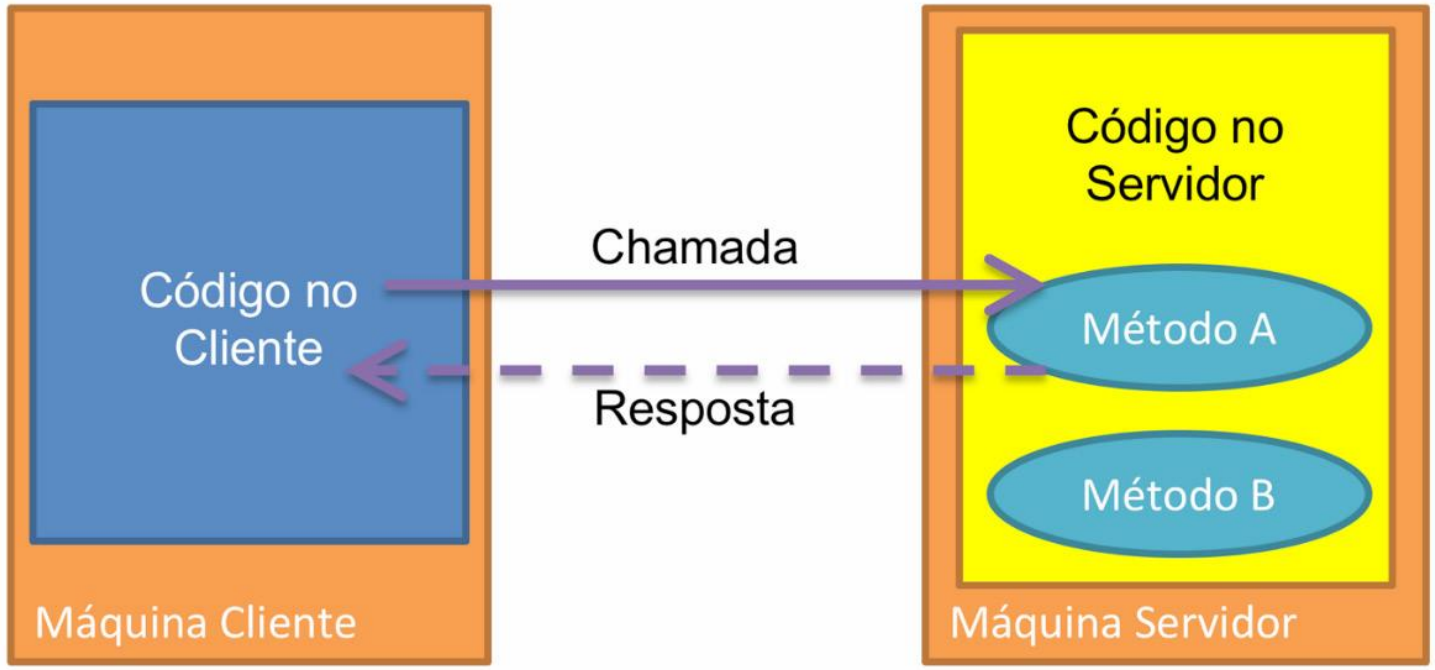
Pyro <https://pythonhosted.org/Pyro4/>
RPyC <https://rpyc.readthedocs.io/en/latest/>

- JavaScript:

Jason <https://www.npmjs.com/package/jayson>

Dnode https://www.npmjs.com/package/dnode

Como pode-se imaginar, cada um desses frameworks possui suas particularidades para implementá-lo, mas a maioria deles possui os componentes ilustrados na Figura 4.13.



Para consolidar o entendimento do RPC, nós iremos realizar uma aplicação simples, utilizando o JRMI e, consequentemente, a linguagem Java. Nossa aplicação de exemplo será constituída de apenas um método remoto que será executado pelo cliente (embora, é claro, poderiam ser implementados mais métodos, conforme a necessidade). Esse método irá receber um nome, em letras minúsculas, e então retornará o mesmo nome, em letras maiúsculas.

Primeiramente, devemos criar uma interface onde iremos declarar os métodos que serão implementados no servidor e, desta forma, que poderão ser executados pelo cliente. Trata-se de uma interface comum, daquelas que você provavelmente deve ter criado em alguma disciplina referente à Orientação a Objetos com Java. A Figura 4.14 apresenta um exemplo de código para essa interface em Java.

Figura 4.14 | Exemplo de código “Interface.java”

```java
import java.rmi.Remote;

public interface Interface extends Remote {

    public String converterParaMaiuscula(String palavra) throws Exception;

}
```

Fonte: elaborada pelo autor.

Como essa interface será implementada para chamadas remotas, através da biblioteca JRMI, devemos explicitamente informar que a mesma irá herdar as propriedades da Classe “Remote”, do Java.

Além disso, como a execução desse método, quando implementado no servidor, será uma operação via rede, devemos alertar à Java Virtual Machine (JVM) que sua execução poderá gerar falhas, em tempo de execução, dependendo das condições da rede. Para simplificar o código, ao invés de tratarmos esse erro, simplesmente alertamos da ocorrência dele à JVM, através da instrução “throws Exception”.

Agora devemos escrever o código que será executado no servidor. A Figura 4.15 apresenta um exemplo de código que implementa o método “converterParaMaiuscula(String palavra)” em Java.

Anhanguera

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

public class Servidor extends UnicastRemoteObject implements Interface {

    public static void main(String[] args) {
        try {
            System.setProperty("java.rmi.server.hostname", "10.116.201.48");
            LocateRegistry.createRegistry(1099);
            Naming.rebind("//10.116.201.48/obj", new Servidor());
        } catch (Exception e) {
            System.err.println("Falha ao iniciar o servidor!");
        }
    }

    public Servidor() throws Exception {
    }

    @Override
    public String converterParaMaiuscula(String palavra) throws Exception {
        return palavra.toUpperCase();
    }
}
```

A implementação do método em si é bastante simples: ele recebe um parâmetro do tipo String, e retorna, para o cliente, essa mesma palavra, em letras maiúsculas, como pode ser visto na linha 22 da Figura 4.14. Mas, para que a chamada de métodos remotos funcione, precisamos de mais algumas informações em nosso código.

A primeira delas é informar qual tipo de comunicação será realizada entre o Servidor e a(s) máquina(s) Cliente: isso é feito através da herança da Classe “UnicastRemoteObject”, apresentada na linha 5 da Figura 4.14. Essa Classe define que a comunicação será do tipo unicast.

Depois, na linha 9 da Figura 4.15, definimos o valor da hostname para o endereço IP da máquina. É importante observar que esse endereço deve ser o endereço IP da máquina onde o código “Servidor.java” será executado. No meu caso, o endereço IP da minha máquina era “10.116.201.48”.

Na linha 10 da Figura 4.15, definimos qual a porta de comunicação será utilizada. Por padrão de mercado, a porta TCP 1099 é utilizada para comunicação através de JRMI.

Por fim, na linha 11 da Figura 4.15, utilizamos o método “rebind”, da Classe “Naming”, para definirmos os parâmetros para criação do objeto que será compartilhado remotamente (e que representa a Classe em si – no meu caso, a Classe “Servidor”) com um endereço IP. Os parâmetros que definimos são:

- Endereço IP: no meu caso, o endereço da minha máquina era 10.116.201.48.
- Nome do objeto que será compartilhado: no meu caso escolhi, arbitrariamente, o nome “obj”.
- Objeto da Classe em questão: no meu caso, utilizei um construtor “vazio” para instanciar um objeto da minha Classe “Servidor”, como pode ser visto nas linhas 17 e 18 da Figura 4.15.

É importante se atentar a esses parâmetros pois, caso queira rodar esse código, provavelmente você precisará alterar essas informações. Como por exemplo o parâmetro do endereço de IP, a execução do programa foi feita em minha máquina, portanto utilizei o endereço de IP 10.116.201.48, o mesmo deve ser alterado para o endereço da máquina que você estiver utilizando para a execução.

Por fim, basta escrevermos o código que será executado na(s) máquina(s) Cliente. A Figura 4.16 apresenta um exemplo de código para esse código.

```
1  import java.rmi.Naming;
2  import java.util.Scanner;
3
4  public class Cliente {
5
6      public static void main(String[] args) throws Exception {
7
8          Interface objetoRemoto = (Interface) Naming.lookup("//10.116.201.48/obj");
9
10         System.out.println(objetoRemoto.converterParaMaiuscula(
11                                         new Scanner(System.in).nextLine())
12                             );
13     }
14 }
```

Anhanguera

No Cliente, a parte principal é “puxarmos” a referência do objeto da Classe “Servidor” para Classe Cliente, ou seja, para a máquina local. Isso é feito através do método “lookup”. Note que esse método recebe um único parâmetro, do tipo String, e ele deve ser idêntico às informações de endereço IP e nome do objeto que foram inseridas o código do Servidor. Feito isso, já podemos utilizar o objeto como se existisse localmente, dentro do código Cliente, através do nome atribuído neste código (eu escolhi o nome “objeto-Remoto”) para chamar e utilizar o método “converterParaMaiúscula”. Se você seguiu as instruções apresentadas até aqui corretamente, ao inserir uma palavra (ou frase) na máquina Cliente, essa palavra (ou frase) será impressa no console, em letras maiúsculas.

Utilizamos o termo RPC quando queremos nos referir a um tipo de comunicação entre máquinas. Esse termo, tecnicamente, é apenas uma das formas de comunicação entre máquinas com alta granularidade, conforme observa Coulouris et al. (2013), sendo, inclusive, a mais antiga delas (porém ainda muito utilizada). Ainda conforme Coulouris, existem três modelos de comunicação entre máquinas: RPC, RMI e MOM.

Assinale a alternativa que contém o significado correto das siglas RPC, RMI e MOM.

a) Remote Procedure Call, Remote Midleware Invocation e Message Oriented Method.

b) Remote Process Call, Remote Method Invocation e Message Oriented Midleware.

c) Remote Procedure Call, Remote Midleware Invocation e Message Oriented Midleware.

d) Remote Process Client, Remote Minute Invite e Message Over Match.

e) Remote Procedure Call, Remote Method Invocation e Message Oriented Midleware.

Dependendo da linguagem de programação a qual o desenvolvedor tem mais familiaridade, existem vários frameworks que podem ser adotados para implementação do RPC. Podemos apontar pelo menos um tipo framework para cada uma das
linguagens mais populares atualmente.
Nesse contexto, analise as afirmativas abaixo e identifique quais delas são verdadeiras ou falsas:
( ) O JRMI e o JMS são frameworks de implementação do RPC através da linguagem Java.
( ) O .NET Remoting e o Dnode são frameworks de implementação do RPC através da linguagem C#.
( ) O Pyro e o RPyC são frameworks de implementação do RPC através da
linguagem Python.
( ) A linguagem JavaScript não possui nenhum framework de implementação do RPC.
Assinale a alternativa que representa a sequência CORRETA.
a) V - V - V - F.
b) V - V - V - V.
c) V - F - V - F.
d) V - F - F - V.
e) F - V - F - V.