UMA BREVÍSSIMA INTRODUÇÃO À JAVA RMI

MCTA025-13 - SISTEMAS DISTRIBUÍDOS

Emilio Francesquini e Fernando Teubl

30 de junho de 2018

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

DISCLAIMER

- Estes slides foram preparados para o curso de **Sistemas Distribuídos na UFABC**.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- Estes slides foram adaptados com base no material disponível em
  - `https://docs.oracle.com/javase/tutorial/rmi/`
  - `https://docs.oracle.com/javase/8/docs/platform/rmi/spec/rmiTOC.html`
  - `https://www.tutorialspoint.com/java_rmi/index.htm`

JAVA RMI

- Java **R**emote **M**ethod **I**nvocation
- Mecanismo que permite que JVMs (Java Virtual Machine) (*i.e.* processos baseados em JVMs) possam se comunicar através da chamada de métodos remotos
- Facilita a comunicação entre os participantes de um sistema distribuído
- SDs criados com JRMI são tipicamente organizados como cliente/servidor mas nada impede que uma aplicação específica adote outras arquiteturas
- Aplicações distribuídas baseadas nesta tecnologia são frequentemente chamadas de sistemas de objetos distribuídos.

ARQUITETURA DE UMA APLICAÇÃO RMI

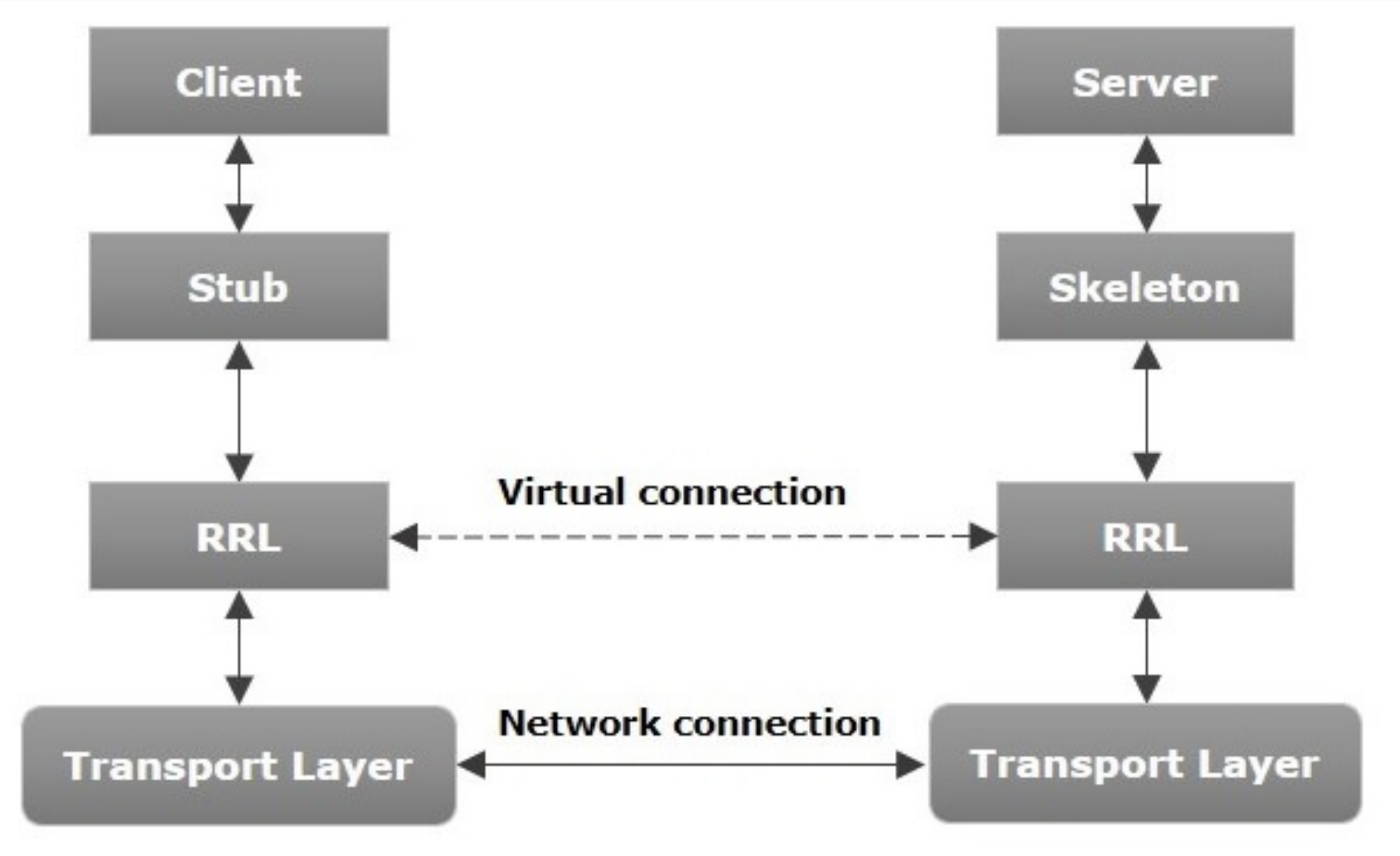

Figura: `https://www.tutorialspoint.com/java_rmi`

ARQUITETURA DE UMA APLICAÇÃO RMI

- Transport Layer Esta camada faz a conexão entre o cliente e o servidor. Suas responsabilidades incluem criar, manter e descartar conexões.
- Stub Um stub é uma representação (como objeto local) de um objeto remoto. Ele atua como um *proxy* do objeto real na JVM do cliente para isto funcionando como intermediário na comunicação entre o programa cliente e o programa servidor.
- Skeleton Localizado no servidor é o responsável por receber as requisições elaboradas pelos stubs dos clientes, repassá-las para o objeto apropriado e enviar a resposta (caso exista) para o cliente.
- RRL(Remote Reference Layer) Esta camada se encarrega de controlar as referências aos objetos disponíveis remotamente para resolver, por exemplo, quando um objeto pode ser descartado pelo coletor de lixo.

FUNCIONAMENTO DE UMA APLICAÇÃO RMI

- Quando um **cliente faz uma chamada** para um método remoto ela é **tratada pelo Stub que a repassa para o RRL**
- Quando o **RRL-Cliente recebe a requisição do Stub** ele **entra em contato com o RRL-Servidor** (através da *Transport Layer*)
- Quando o **RRL-Servidor** recebe a requisição ele a **repassa ao Skeleton** que por sua vez faz a **requisição para o objeto remoto no servidor**.
- A **resposta**, caso exista, é então enviada seguindo-se o mesmo **caminho (reverso) até o cliente**.

EMPACOTAMENTO E DESEMPACOTAMENTO DE OBJETOS

- Quando uma chamada a um método de um objeto remoto que receba parâmetros é feita, é preciso fazer o empacotamento (*marshalling*) destes parâmetros para que sejam transmitidos pela rede até o servidor.
- O servidor, por sua vez, precisa fazer o desempacotamento (*unmarshalling*) desses parâmetros para repassá-los ao objeto.
- O mesmo processo é necessário para devolver ao cliente o resultado da chamada do método caso ele devolva um valor.
- Podem ser empacotados/desempacotados: tipos primitivos e objetos que implementem a interface `java.io.Serializable`

LOCALIZANDO OBJETOS REMOTOS - RMI REGISTRY

- O RMI Registry mantém um catálogo com os objetos remotos colocados a disposição
- Toda vez que um processo deseja colocar um objeto a disposição dos clientes ele o registra (*bind*) no RMI Registry
- Quando um cliente deseja encontrar um objeto remoto ele consulta (*lookup*) o RMI Registry para obter uma referência àquele objeto (representada pelo stub)

RMI REGISTRY

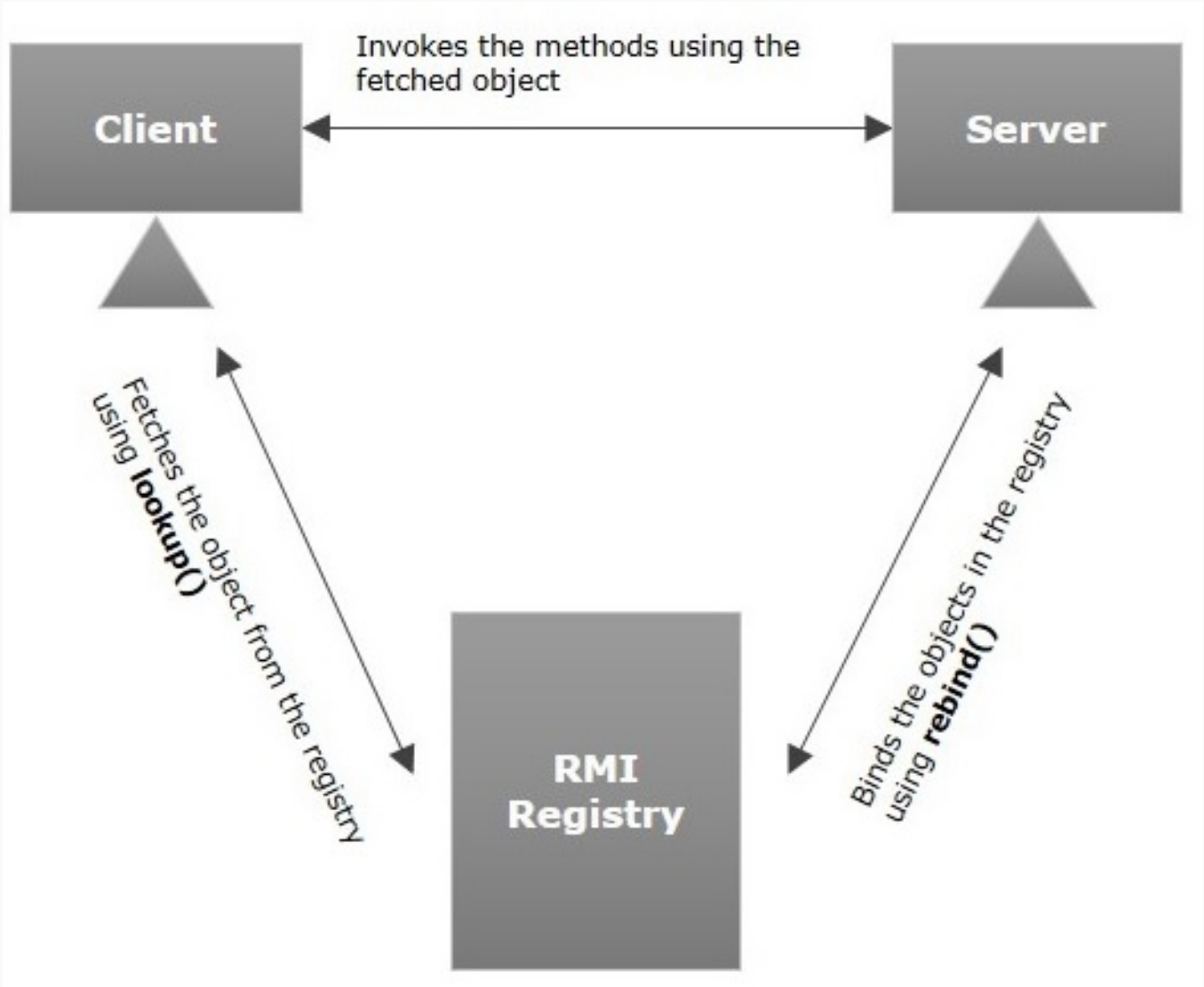

Figura: `https://www.tutorialspoint.com/java_rmi`

DESENVOLVENDO UMA APLICAÇÃO COM RMI

- Como qualquer outra aplicação em Java, uma aplicação com RMI é baseada em classes e interfaces
  - Interfaces declaram métodos, classes implementam métodos
- Objetos cujos métodos podem ser chamados entre JVMs distintas são chamados objetos remotos.
- Numa aplicação distribuída alguns objetos da aplicação podem ser locais e outros remotos.
- **Para se tornar um objeto remoto um objeto** precisa implementar uma interface remota
  - Uma interface remota estende a interface `java.rmi.Remote`
  - Todos os métodos de uma interface remota devem lançar a exceção `java.rmi.RemoteException`

DESENVOLVENDO UMA APLICAÇÃO RMI

Os passos básicos abaixo descrevem o que é preciso ser feito para criar uma aplicação RMI

- Definir as interfaces remotas da aplicação
  - Quais são objetos e os métodos relevantes para a aplicação
- Desenvolver a classe contendo a implemtação do objeto
  - É comum pelo menos parte do sistema que deseja-se tornar distribuído já existir, neste caso se faz necessário adaptá-lo para ficar compatível com a interface remota definida no passo anterior
- Desenvolver o servidor
- Desenvolver o cliente

VISÃO GERAL

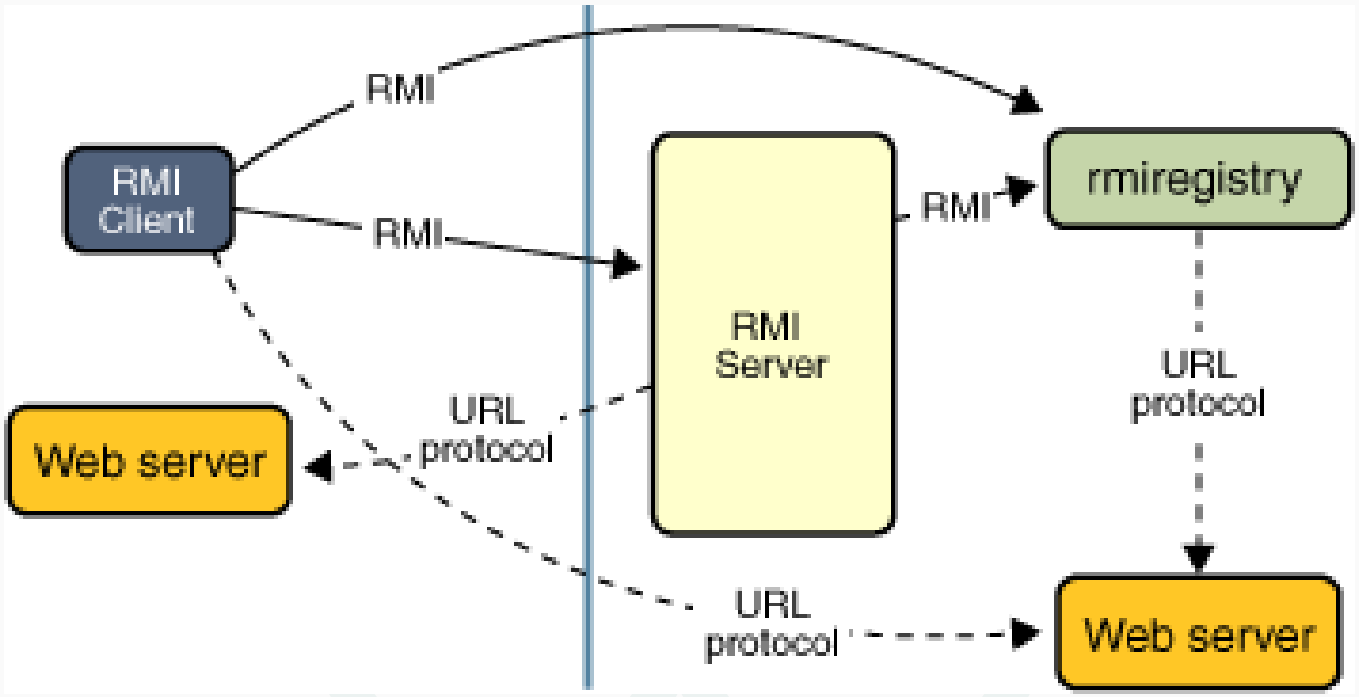

Figura: `https://docs.oracle.com/javase/tutorial/rmi/`

EXEMPLO: CALCULADORA SIMPLES

- Como exemplo vamos desenvolver uma calculadora simples, que aceita as 4 operações aritméticas básicas
- Apesar de não ser necessário, por motivos didáticos vamos fazer com que os parâmetros não sejam tipos primitivos (`int`,`float`,`double`) mas um tipo `Numero` que vamos criar

A INTERFACE NÚMERO

```java
public interface Numero extends java.io.Serializable {
    double getValor();
}
```

A CLASSE NÚMERO

```
1  public class NumeroImpl implements Numero {
2
3      private double num;
4
5      public NumeroImpl (double val) {
6          num = val;
7      }
8
9      public double getValor() {
10         return num;
11     }
12
13 }
```

A INTERFACE CALCULADORA

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

public interface Calculadora extends Remote {
    public Numero soma (Numero a, Numero b)
        throws RemoteException;

    public Numero subtrai (Numero a, Numero b)
        throws RemoteException;

    public Numero multiplica (Numero a, Numero b)
        throws RemoteException;

    public Numero divide (Numero a, Numero b)
        throws RemoteException, DivisaoPorZeroException;
 }
```

A CLASSE CALCULADORA

```java
public class CalculadoraImpl implements Calculadora {

    public Numero soma (Numero a, Numero b) {
        return new NumeroImpl (a.getValor() + b.getValor());
    };

    public Numero subtrai (Numero a, Numero b) {
        return new NumeroImpl (a.getValor() - b.getValor());
    };

    public Numero multiplica (Numero a, Numero b)  {
        return new NumeroImpl (a.getValor() * b.getValor());
    };

    public Numero divide (Numero a, Numero b)
        throws DivisaoPorZeroException  {
        if (b.getValor() == 0) throw new DivisaoPorZeroException();
        return new NumeroImpl (a.getValor() / b.getValor());
    };

}
```

O SERVIDOR

```java
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.server.UnicastRemoteObject;
public class ServidorCalculadora {
  public static void main(String args[]) {
    try {
      //Crio o objeto servidor
      CalculadoraImpl calc = new CalculadoraImpl();
      //Criamos o stub do objeto que será registrado
      Calculadora stub = (Calculadora)UnicastRemoteObject
          .exportObject(calc, 0);
      //Registra (binds) o stub no registry
      Registry registry = LocateRegistry.getRegistry();
      registry.bind("calculadora", stub);
      System.out.println("Servidor iniciado.");
    } catch (Exception e) {
          System.err.println("Ocorreu um erro no servidor: " +
                             e.toString());
    }
  }
}
```

O CLIENTE - PARTE 1

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ClienteCalculadora {
  public static void main(String[] args) {
    try {

        // Localiza o registry. É possível usar endereço/IP porta
      Registry registry = LocateRegistry.getRegistry(null);
      // Consulta o registry e obtém o stub para o objeto remoto
      Calculadora calc = (Calculadora) registry
          .lookup("calculadora");
      // A partir deste momento, cahamadas à Caluladora podem ser
      // feitas como qualquer chamada a métodos
```

O CLIENTE - PARTE 2

```
16          Numero num1 = new NumeroImpl(3);
17          Numero num2 = new NumeroImpl(4);
18          //Aqui são feitas diversas chamadas remotas
19          Numero soma = calc.soma(num1, num2);
20          Numero sub = calc.subtrai(num1, num2);
21          Numero mult = calc.multiplica(num1, num2);
22          Numero div = calc.divide(num1, num2);
23          System.out.println("Resultados obtidos do servidor:" +
24                             "\n\t+:" + soma.getValor() +
25                             "\n\t-:" + sub.getValor()  +
26                             "\n\t*:" + mult.getValor() +
27                             "\n\t/:" + div.getValor());
```

O CLIENTE - PARTE 3

```
29          try {
30              calc.divide(new NumeroImpl(1), new NumeroImpl(0));
31          } catch (DivisaoPorZeroException e) {
32              System.out.println(
33                "Tentou dividir por zero! Esta é uma exceção remota.");
34          }
35
36      } catch (Exception e) {
37          System.err.println("Ocorreu um erro no cliente: " +
38                             e.toString());
39      }
40    }
41  }
```

EXECUTANDO A APLICAÇÃO - RMI REGISTRY

É preciso iniciar o *registry* e então o servidor para que ele registre a calculadora.

```
$ rmiregistry
```

Não é dada nenhuma saída, a menos que outro registry já esteja em execução. Neste caso:

```
$ rmiregistry
java.rmi.server.ExportException: Port already in use: 1099; nested exc
        java.net.BindException: Address already in use (Bind failed)
        ...
```

EXECUTANDO A APLICAÇÃO

Em um outro terminal, inicie o servidor:

```
$ java ServidorCalculadora
Servidor iniciado.
```

E finalmente o cliente:

```
$ java ClienteCalculadora
Resultados obtidos do servidor:
        +:7.0
        -:-1.0
        *:12.0
        /:0.75
Tentou dividir por zero! Esta é uma exceção remota.
$
```

EXERCÍCIO

Implemente um servidor de chat com suporte a múltiplos clientes (a exemplo do que foi feito na Aula 1) utilizando JRMI.