

Fundamentos de Objetos Remotos

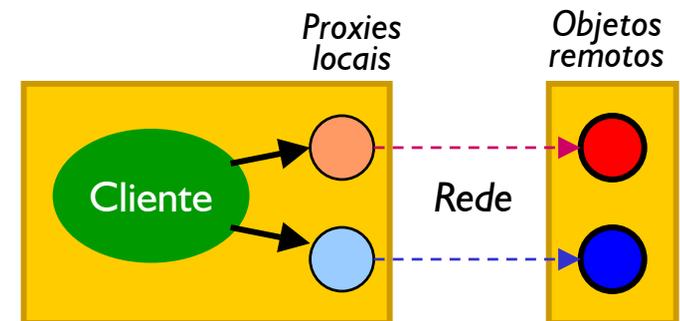
Helder da Rocha
www.argonavis.com.br

Sobre este módulo

- *Este módulo tem como objetivo dar uma visão geral, porém prática, da criação e uso de objetos remotos em Java com RMI*
- *Para permitir a demonstração de uma aplicação simples, vários conceitos importantes, essenciais em aplicações RMI reais foram omitidos, como*
 - *Necessidade de implantar um gerente de segurança*
 - *Necessidade de divulgar um codebase (Classpath distribuído)*
- *Para maiores detalhes, consulte as fontes de referência para este curso*

O que são objetos remotos

- *Objetos remotos são objetos cujos métodos podem ser chamados remotamente, como se fossem locais*
 - *Cliente precisa, de alguma forma, localizar e obter uma instância do objeto remoto (geralmente através de um proxy intermediário gerado automaticamente)*
 - *Depois que o cliente tem a referência, faz chamadas nos métodos do objeto remoto (através do proxy) usando a mesma sintaxe que usaria se o objeto fosse local*
- *Objetos remotos abstraem toda a complexidade da comunicação em rede*
 - *Estende o paradigma OO além do domínio local*
 - *Torna a rede transparente*



Como implementar objetos remotos

- Para usar objetos remotos, é preciso ter uma infraestrutura que cuide da geração das classes, comunicação e outros detalhes
- Há duas soluções disponíveis para implementar objetos remotos em Java
 - **OMG CORBA**: requer o uso, além de Java, da linguagem genérica **OMG IDL**, mas suporta integração com outras linguagens
 - **Java RMI**: para soluções 100% Java
- As duas soluções diferem principalmente na forma de implementação
- Em ambas, um programa cliente poderá **chamar um método em um objeto remoto** da mesma maneira como faz com um método de um objeto local

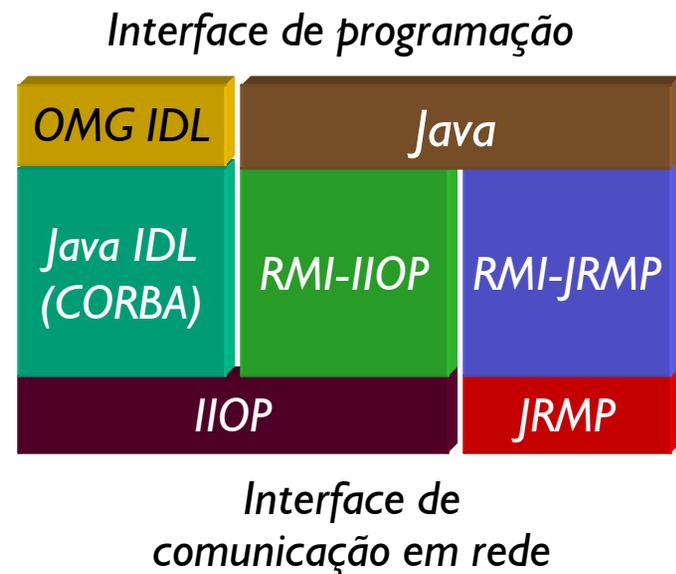
Interface de programação



Interface para comunicação em rede

Objetos remotos com Java RMI

- Java RMI (**R**emote **M**ethod **I**nvocation) pode ser implementado usando protocolos e infraestrutura próprios do Java (JRMP e RMI Registry) ou usando IIOP e ORBs, próprios do CORBA
- **JRMP** - **J**ava **R**emote **M**ethod **P**rotocol
 - Pacote `java.rmi` - **RMI básico**
 - Ideal para aplicações 100% Java.
- **IIOP** - **I**nternet **I**nter-**O**RB **P**rotocol
 - Pacote `javax.rmi` - **RMI sobre IIOP**
 - Ideal para ambientes heterogêneos.
- A forma de desenvolvimento é similar
 - Há pequenas diferenças na geração da infraestrutura (proxies) e registro de objetos
- RMI sobre IIOP permite **programação Java RMI** e **comunicação em CORBA**, viabilizando integração entre Java e outras linguagens sem a necessidade de aprender OMG IDL



RMI: funcionamento básico

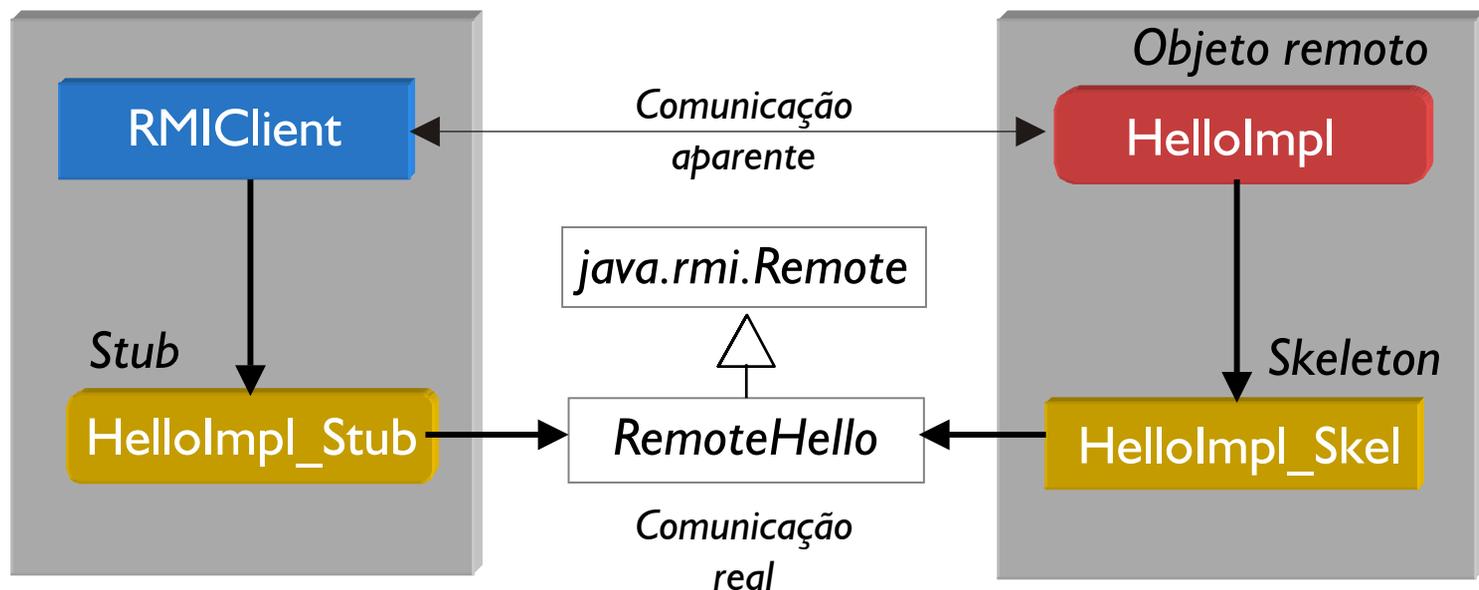
- Um objeto remoto **previamente registrado** é obtido, através de servidor de nomes especial: **RMI Registry**.
 - Permite que os objetos publicamente acessíveis através da rede sejam referenciados através de um **nome**.
- Serviço de nomes: classe **java.rmi.Naming**
 - Método **Naming.lookup()** consulta um servidor de nomes RMI e obtém uma instância de um objeto remoto
- Exemplo (jogo de batalha naval):

```
Territorio mar =  
    (Territorio)Naming.lookup("rmi://gamma/caspio");
```
- Agora é possível chamar métodos remotos de mar:

```
tentativa[i] = mar.atira("C", 9);
```

Arquitetura RMI

- Uma aplicação distribuída com RMI tem acesso transparente ao objeto remoto através de sua **Interface Remota**
 - A "Interface Remota" é uma interface que estende `java.rmi.Remote`
 - A partir da Interface Remota e implementação do objeto remoto o sistema gera objetos (proxies) que realizam todas as tarefas necessárias para viabilizar a comunicação em rede



Padrões de Projeto: Proxy

- A implementação RMI é um exemplo do padrão de projeto chamado **Proxy**
- Proxy é uma solução para situações onde o objeto de interesse está inacessível diretamente, mas o cliente precisa operar em uma interface idêntica
 - A solução oferecida por Proxy é criar uma classe que tenha a mesma interface que o objeto de interesse (implemente a mesma interface Java) e que implemente, em seus métodos, a lógica de comunicação com o objeto inacessível.
 - Em RMI, o proxy é o Stub gerado automaticamente pelo ambiente de desenvolvimento (rmic)

Como usar RMI em 10 passos

- O objetivo deste módulo é oferecer apenas uma *introdução básica a Java RMI*. Isto será feito através de um exemplo simples
 - 1. Definir a interface
 - 2. Implementar os objetos remotos
 - 3. Implementar um servidor para os objetos
 - 4. Compilar os objetos remotos
 - 5. Gerar stubs e skeletons com *rmic*
 - 6. Escrever, compilar e instalar o(s) cliente(s)
 - 7. Instalar o stub no(s) cliente(s)
 - 8. Iniciar o RMI Registry no servidor
 - 9. Iniciar o servidor de objetos
 - 10. Iniciar os clientes informando o endereço do servidor.

I. Definir a interface remota

- Declare todos os métodos que serão acessíveis remotamente em uma interface Java que estenda *java.rmi.Remote*.
 - Todos os métodos devem declarar *throws java.rmi.RemoteException*.
- Isto deve ser feito para cada objeto que será *acessível* através da rede.

```
import java.rmi.*;
public interface Mensagem extends Remote {
    public String getMensagem()
        throws RemoteException;
    public void setMensagem(String msg)
        throws RemoteException;
}
```

2. Implementar os objetos remotos

- Cada objeto remoto é uma classe que estende a classe `java.rmi.server.UnicastRemoteObject` e que implementa a interface remota criada no passo 1.
- Todos os métodos declaram causar `java.rmi.RemoteException` inclusive o construtor, mesmo que seja vazio.

```
import java.rmi.server.*;
import java.rmi.*;

public class MensagemImpl extends UnicastRemoteObject
    implements Mensagem {
    private String mensagem = "Inicial";
    public MensagemImpl() throws RemoteException {}
    public String getMensagem() throws RemoteException {
        return mensagem;
    }
    public void setMensagem(String msg) throws RemoteException {
        mensagem = msg;
    }
}
```

3. Estabelecer um servidor

- Crie uma classe que
 - a) Crie uma instância do objeto a ser servido e
 - b) Registre-a (*bind* ou *rebind*) no serviço de nomes.

```
import java.rmi.*;
public class MensagemServer {

    public static void main(String[] args)
        throws RemoteException {
        Mensagem mens = new MensagemImpl();
        Naming.rebind("mensagens", mens);
        System.out.println("Servidor no ar. "+
            " Nome do objeto servido: '"
                + "mensagens" + "'");
    }
}
```

4. Compilar os objetos remotos

- *Compile todas as interfaces e classes utilizadas para implementar as interfaces Remote*
 - `javac Mensagem.java MensagemImpl.java`

5. Gerar stubs e skeletons

- Use a ferramenta do J2SDK: **rmic**
- Será gerado um arquivo stub
 - **MensagemImpl_stub.class**e um arquivo skeleton
 - **MensagemImpl_skel.class**para cada objeto remoto (neste caso, apenas um)
- RMIC = **RMI C**ompiler
 - Use opção **-keep** se quiser manter código-fonte
 - Execute o **rmic** sobre as implementações do objeto remoto já compiladas:
 - > **rmic** MensagemImpl

6. Compilar e instalar o(s) cliente(s)

- Escreva uma classe cliente que localize o(s) objeto(s) no serviço de nomes (*java.rmi.Naming*)
 - a) Obtenha uma instância remota de cada objeto
 - b) Use o objeto, chamando seus métodos

```
import java.rmi.*;
public class MensagemClient {
    public static void main(String[] args)
        throws Exception {
        String hostname = args[0];
        String objeto = args[1];
        Object obj =
            Naming.lookup("rmi://" + hostname + "/"
                + objeto);
        Mensagem mens = (Mensagem) obj;
        System.out.println("Mensagem recebida: "
            + mens.getMensagem());
        mens.setMensagem("Fulano esteve aqui!");
    }
}
```

7. Instalar o stub no(s) cliente(s)

- *Distribua o cliente para as máquinas-cliente. A distribuição deve conter*
 - *Classe(s) que implementa(m) o cliente (`HelloMensagem.class`)*
 - *Os stubs (`HelloMensagem_stub.class`)*
 - *As interfaces Remote (`Mensagem.class`)*
- *Em aplicações reais, os stubs podem ser mantidos no servidor*
 - *O cliente faz download do stub quando quiser usá-lo*
 - *Para isto é preciso definir algumas propriedades adicionais (omitidas no nosso exemplo simples) como `Codebase` (`CLASSPATH` distribuído), `SecurityManager` e políticas de segurança (`Policy`)*

8. Iniciar o RMI Registry no servidor

- *No Windows*

 - > `start rmiregistry`

- *No Unix*

 - > `rmiregistry &`

 - (O RMI Registry fica "calado" quando está rodando)*

- *Neste exemplo será preciso iniciar o RMIRegistry no diretório onde estão os stubs e interface Remote*

 - *Isto é para que o RMIRegistry veja o mesmo CLASSPATH que o resto da aplicação*

 - *Em aplicações RMI reais isto não é necessário, mas é preciso definir a propriedade `java.rmi.server.codebase` contendo os caminhos onde se pode localizar o código*

9. Iniciar o servidor de objetos

- O servidor é uma aplicação executável que registra os objetos no *RMIRegistry*. Rode a aplicação:
 - > `java MensagemServer`
Servidor no ar. Nome do objeto servido: mensagens
- Neste exemplo será preciso iniciar o servidor no diretório onde estão os stubs e interface *Remote*
 - Isto é para que o *RMIRegistry* veja o mesmo *CLASSPATH* que o resto da aplicação
 - Em aplicações RMI reais isto não é necessário, mas é preciso definir a propriedade `java.rmi.server.codebase` contendo os caminhos onde se pode localizar o código

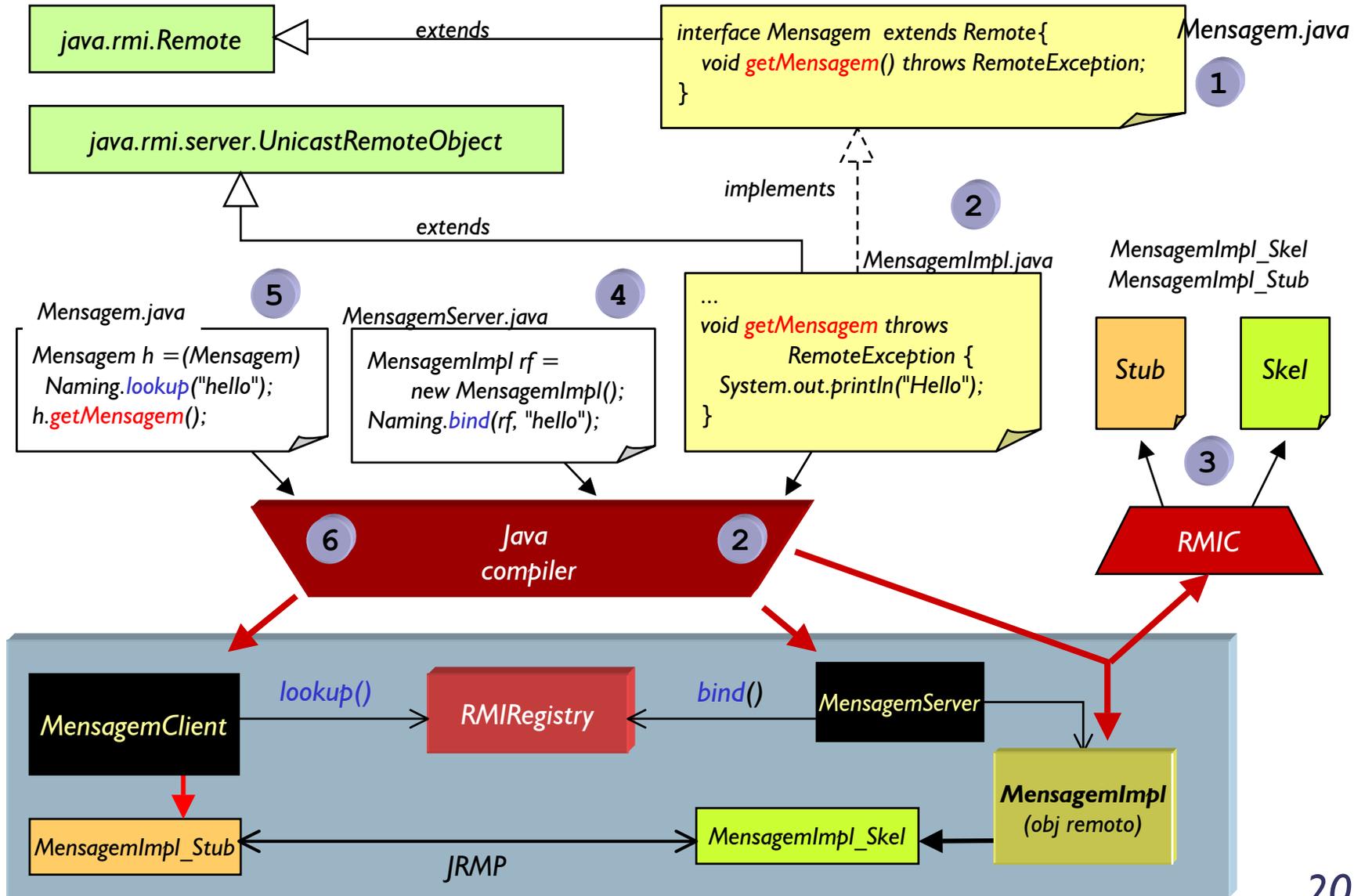
10. Iniciar os clientes

- *Rode o cliente*
 - *Informe o endereço do servidor e objetos a utilizar*
- ```
> java MensagemClient maquina.com.br mensagens
```

## **Exercício**

Implemente os exemplos mostrados, inventando uma mensagem diferente para seu objeto remoto. Use o seu cliente para acessar os objetos remotos registrados nas máquinas de seus colegas.

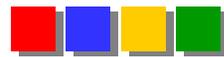
# Resumo



# Curso J100: Java 2 Standard Edition

Revisão 17.0

© 1996-2003, Helder da Rocha  
(helder@acm.org)

 argonavis.com.br