Fundamentos de Objetos Remotos

Helder da Rocha
www.argonavis.com.br

Objetos remotos

- *Há duas soluções de objetos remotos em Java*
  - *OMG CORBA*
  - *Java RMI*
- *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*
- *Java RMI usa interfaces Java como fachada para os objetos remotos*
- *CORBA usa OMG IDL (Interface Definition Language)*

Objetos remotos com Java RMI

- Java RMI pode ser implementado usando protocolos e infraestrutura próprios ou usando IIOP e ORBs
- JRMP - Java Remote Method Protocol + RMI Registry
  - Pacote java.rmi
  - Ideal para aplicações 100% Java.
- IIOP - Internet Inter-ORB Protocol + ORBs
  - Pacote javax.rmi
  - Ideal para ambientes heterogêneos.
- A forma de desenvolvimento é semelhante.
- Principais diferenças: geração da infraestrutura (stubs, etc.) e registro de objetos remotos

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):

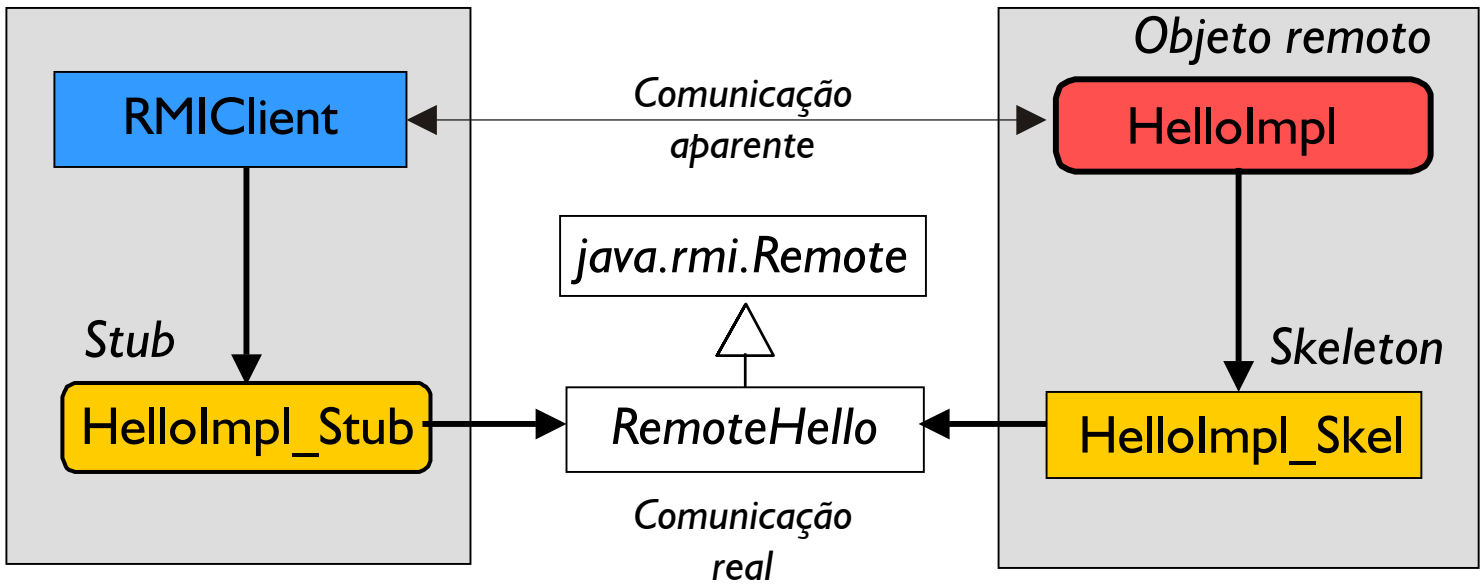
```
Territorio mar =
  (Territorio)Naming.lookup("rmi://gamma/caspio");
```

- 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 Remote*
  - *A partir de sua interface Remote e implementação do objeto remoto são gerados objetos que realizam todas as tarefas necessárias para viabilizar a comunicação em rede*



Como usar: passo-a-passo

- O objetivo deste módulo é oferecer apenas uma introdução básica a 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.

1. Definir a interface

- *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 RemoteHello 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 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 HelloImpl extends UnicastRemoteObject
                                        implements RemoteHello {
    private String mensagem = "Inicial";
    public HelloImpl() 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 obtenha uma instância do objeto a ser servido.*

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

    public static void main(String[] args)
                                throws RemoteException {
            RemoteHello hello = new HelloImpl();
            Naming.rebind("mensagens", hello);
            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 HelloRemote.java HelloImpl.java`

5. Gerar stubs e skeletons

- Use **rmic**
- Será gerado um arquivo stub
  - HelloImpl_stub.class

  e um arquivo skeleton
  - HelloImpl_skel.class
- RMIC - RMI Compiler
  - Use opção -keep se quiser manter código-fonte
  - Para executar:

```
> rmic HelloImpl
```

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

- *Escreva uma classe cliente que localize o(s) objeto(s)*
  - *Obtenha uma instância remota de cada objeto*
  - *Chame os objetos remotos de cada objeto*

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

** Este cliente foi simplificado (não tem SecutiryManager) para facilitar a demonstração.*

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

- Compile e distribua o cliente para as máquinas-cliente
  - A(s) classe(s) que implementa(m) o cliente
  - Os stubs
  - As interfaces Remote
- Geralmente os stubs são mantidos no servidor
  - O cliente pode fazer download do stub e começar a usá-lo
  - É preciso definir propriedades adicionais (omitidas neste exemplo simples)
  - Aplicações RMI típicas usam
    - Codebase: "CLASSPATH distribuído"
    - SecurityManager
    - Arquivo com políticas de segurança

8. Iniciar o RMI Registry no servidor

- No Windows
  - start rmiregistry
- No Unix
  - rmiregistry&
- 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. Rode a aplicação:*

```
c:\> java HelloServer
   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 os clientes e informe o endereço do servidor e objetos a utilizar*

```
c:\> java HelloClient maquina.com.br mensagens
```

Exercícios

- *1. Crie uma aplicação distribuída para objetos Cartaz.*
  - *Um Cartaz tem dois Color: corDeFundo e corDoTexto, e uma String texto*
  - *Um cliente pode*
    - *Criar um novo objeto Cartaz e gravá-lo no servidor*
    - *Ler um objeto Cartaz do servidor e exibi-lo pintando a área de trabalho com a cor de fundo, e imprimindo o texto, no centro, em fonte 36pt SansSerif com a cor do texto.*
  - *Observações:*
    - *Cartaz deve ser Serializable*
    - *O servidor deve imprimir na tela que está no ar e qual objeto está servindo.*

Exercícios (2)

- 2. Estenda o software da biblioteca para que funcione com objetos remotos
  - Crie uma interface remota para todos os métodos da fachada Biblioteca
  - Implemente um objeto remoto que chame os métodos da fachada (Biblioteca)
  - Crie um cliente remoto que implemente RepositorioDados (RepositorioDadosRMI)
  - Gere os stubs, skeletons e monte a aplicação RMI
- Para detalhes de como configurar a aplicação para uso em rede (com policy, SecurityManager, etc.)
  - Veja código exemplo em cap20/ e arquivo javanet_6.ppt (slides do curso avançado sobre RMI)