Java RMI – Remote Method Invocation

Prof. Tiago Garcia de Senna Carneiro

Sistemas Distribuídos - 2006

Conteúdo

- Passos para implementar um aplicação RMI
- Executando e compilando uma aplicação RMI
- Exemplo: gerenciamento de contas bancárias
- Configuração RMI
- Applets RMI

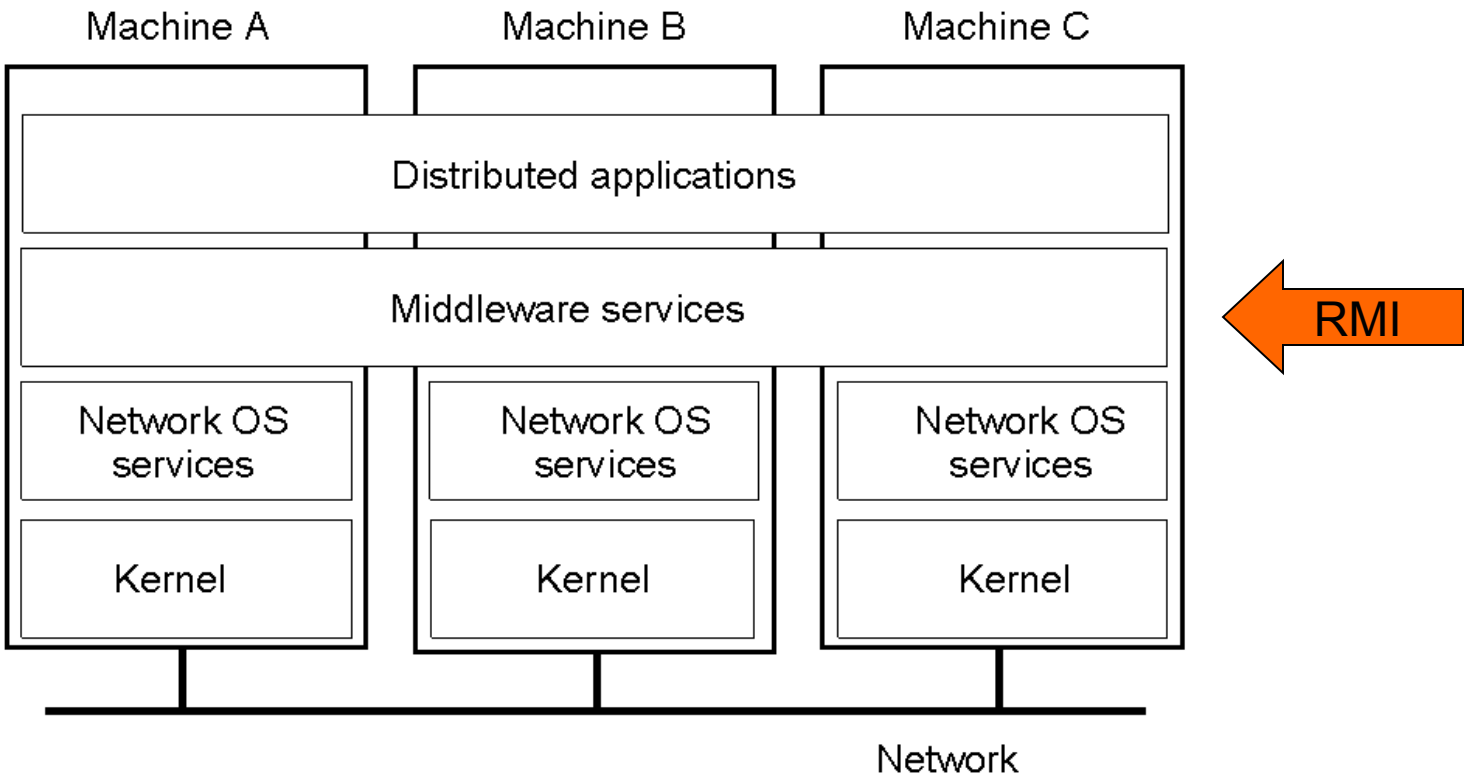
Introdução

- Idéia básica:
  - Distribuir objetos por uma rede de computadores para tirar vantagem do poder agregado de processamento e compartilhar recursos.
  - O usuário requer a instância de uma classe usando uma sintaxe de uma URL

    rmi:\\servidor:porto\nome
  - Usuários utilizam os objetos como se eles fossem objetos locais
    - Conexões de rede são abertas e fechadas automaticamente
    - O mecanismo de “serialização” de JAVA permite a passagem de estruturas de dados complexas como parâmetro pela rede sem que seja necessário desenvolver um analisador sintático para empacotá-los e desempacotá-los

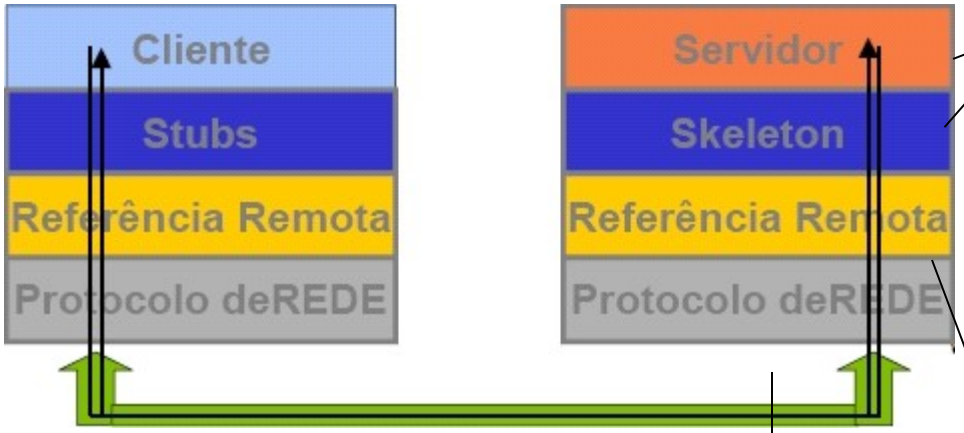
Arquitetura RMI

- O arquitetura RMI implementa uma idéia simples: o *middleware*.



Arquitetura RMI

- Do lado cliente : stub
- Do lado servidor: skeleton



Aplicação

Skeleton: despacha chamadas para a implementação real do objeto remoto

Stub: invoca método remoto

Camada de Referência Remota

- gerencia a comunicação entre stubs e skeletons
- Gerencia as referência a objetos remotos
- Gerencia estratégias de reconexão caso o objeto fique indisponível

Camada de Transporte

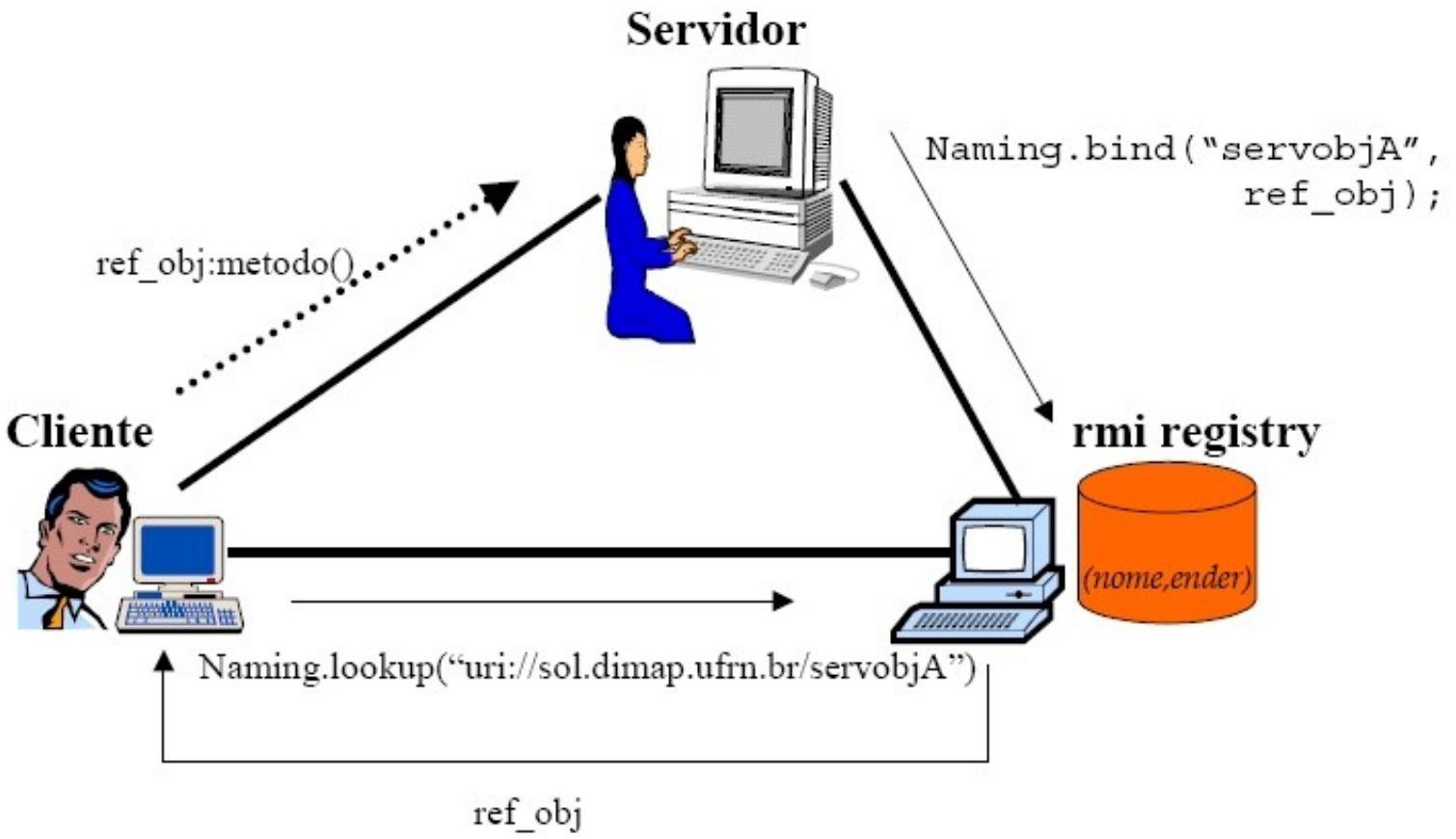
- configuração do soquete utilizado para comunicação entre o cliente e o servidor
- Registro do servidor RMI (lado servidor)
- localização do servidor RMI do objeto remoto requisitado (lado cliente)

Camada Stub/Skeleton

- Stub:
  - Empacota identificador do objeto remoto
  - Empacota o identificador do método
  - Ordena e empacota parâmetros
  - Envia pacote para o skeleton

- Skeleton:
  - Desempacota e analise parâmetros
  - Executa método e obtém valor de retorno ou executa uma exeção
  - Ordena e empacota valor de retorno
  - Envia pacote para o cliente

RMI - Registry



Java.rmi.*

- Classe Naming :
  - classe utilizada por servidores e cliente para comunicação com o registro do servidor (RMIRegistry).

- Servidor:
  - *bind e rebind:* registrar suas implementações de objetos

Se o nome estiver em uso, ocorre uma exceção.

```
// Liga a instância do objeto ao registro
System.out.println("Banco RMI: ligando a ao nome - gerenteBanco");
Naming.rebind("gerenteBanco", gc);
System.out.println("Servidor gerenteBanco pronto.");
```

- Cliente:
  - Lookup: obter uma referência ao objeto remoto.

```
// Obtem uma instância para o objeto remoto
System.out.println("Clinte RMI: pesquisando nome - gerenteBanco");
String url = new String("rmi://"+args[0]+"/gerenteBanco");
GerenteConta gc = (GerenteConta) Naming.lookup(url);
```

Caso de estudo: Sistema Bancário

- Objetos remotos:
  - Várias conta bancária e
  - Um gerente de contas

- Serviços oferecidos pelo sistema:
  - depósitos
  - saque e
  - verificações do saldo.

Construindo uma aplicação RMI – Passo a Passo –

- Uso Padrão de desenvolvimento de software: Method Factory
  - permite que um objeto controle a criação de outros objetos
- Interfaces Java:
  - interface Conta extends Remote
  - interface GerenteConta extends Remote

Construindo uma aplicação RMI – Passo a Passo –

0. Tenha certeza que as variáveis de sistema PATH e CLASSPATH estejam configurada corretamente.(Veja arquivo rmi.bat)

set path=%path%;c:\jdk1.3.0_02\bin
set classpath=%classpath%;c:\meuDiretorio\Java\rmi

1. Defina os objetos remotos a serem operados como interfaces Java.

2. Crie as classes de implementação das interfaces.

3. Compile a interface e as classes de implementação.

4. Crie as classes de *esqueleto* e *stub* utilizando o comando *rmic* das classes de implementação.

rmic –vcompat ContaImpl
rmic –vcompat GerenteContaImpl

Construindo uma aplicação RMI
– Passo a Passo –

5. Crie e compile um aplicativo servidor para administrar as implementações.

6. Crie e compile um aplicativo cliente para acessar os objetos remotos

7. Inicie o RMIregistry a partir de um diretório que não contenha arquivos *.class

start rmiregistry

8. Execute o servidor

start java -Djava.security.policy=servidor.policy ServidorBanco

9. Teste o cliente.

java -Djava.security.policy=cliente.policy ClienteBanco localhost Jose

Interface Conta

```
import java.rmi.*;
import banco.DinheiroFalsoException;


// A interface da classe Conta
public interface Conta extends Remote {

    // Consultar o saldo da conta
    public float getSaldo() throws RemoteException;

    // Depositar dinheiro na conta - lança uma execeção se o valor for 0
    // ou um número negativo
    public void deposito( float dinheiro )
        throws DinheiroFalsoException, RemoteException;

    // Sacar dinheiro na conta - lança uma execeção se o valor do saque
    // exceder o saldo da conta
    public void saque(float dinheiro)
        throws DinheiroFalsoException, RemoteException;

}
```

Interface GerenteConta

```
import java.rmi.*;
import Conta;
import banco.DinheiroFalsoException;


public interface GerenteConta extends Remote {

    // Cria uma nova conta ou retorna a instância atual do objeto Conta caso
    // ela já exista.
    public Conta abrir( String nome, float saldoInicial )
        throws DinheiroFalsoException, RemoteException;
}
```

Exceção DinheiroFalso

```
package banco;

public class DinheiroFalsoException extends Exception {
    String motivo;

    public DinheiroFalsoException( String motivo ) {
        this.motivo = motivo;
    }
}
```

Objeto Conta

Classe UnicasRemoteObject: implementa um objeto remoto cuja referência é válida durante o tempo de vida do servidor.

Classe Activatable: objetos remotos que requerem accessos persistentes no tempo e que podem ser ativados pelo sistema.

```
public class ContaImpl
     extends UnicastRemoteObject
     implements Conta {

     private float saldo  = 0;

     public ContaImpl( float novoSaldo ) throws RemoteException {
          saldo = novoSaldo;
     }

     // Consultar o saldo da conta
     public float getSaldo() throws RemoteException {
         return saldo;
     }

     // Depositar dinheiro na conta - lança uma execeção se o valor for 0
     // ou um número negativo
     public void deposito( float dinheiro ) ..

     // Sacar dinheiro na conta - lança uma execeção se o valor do saque
     // exceder o saldo da conta
     public void saque(float dinheiro)..

}
```

Objeto Gerente Conta

```
public class GerenteContaImpl
    extends UnicastRemoteObject
    implements GerenteConta {

    // Armazenamento local dos nomes das Contas
    private static Vector contas = new Vector();

    // Esse construtor vazio é necessário para criar uma instância dessa
    // classe no servidor
    public GerenteContaImpl( ) throws RemoteException {
    }

    // Cria uma nova conta ou retorna a instância atual do objeto Conta caso
    // ela já exista.
    public Conta abrir( String nome, float saldoInicial )

}

class ContaInfo {
    String name;
    ContaImpl conta = null;
}
```

Instalando a apliacação RMI

- Os arquivos *stubs* precisam ser colocados em um servidor HTTP para douwnload
  - No Java 2, o protocolo RMI 1.2 dispensa o *skeleton*
- Os clientes devem instalar um RMISecurityManager para carregar classes RMI remotamente

System.setSecurityManager(new RMISecurityManager( ));

- Os clientes precisam de um arquivo de *politica de acesso (policy file)* para conectarem se aos RMIRegistry e servidores HTTP

O arquivo de política de acesso para o CLIENTE

```
grant {
  // rmihost - RMI registry and the server
  // webhost - HTTP server for stub classes
  permission java.net.SocketPermission
    "rmihost:1024-65535", "connect";
  permission java.net.SocketPermission
    "webhost:80", "connect";
};
```

O *servidor* RMI comunica-se com o *cliente* e com o *servidor de nome* através de um porto escolhido de forma aleatória

O arquivo de política de acesso para o SERVIDOR

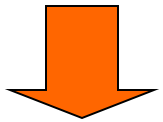
```
grant
{
permission java.net.SocketPermission "*:1099","accept,connect,listen";
};
```

O Servidor RMI

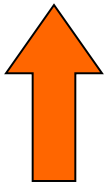
```
public class ServidorBanco {
    public static void main( String args[]){

        // Cria e instala o gerente de segurança
       //   System.setSecurityManager( new RMISecurityManager());
        try {
            // cria a instância do objeto para registro
            System.out.println("BANCO: crinando uma GerenteContaImpl");
            GerenteContaImpl gc = new GerenteContaImpl();

            // Liga a instância do objeto ao registro
            System.out.println("Banco RMI: ligando a ao nome - gerenteBanco");
            Naming.rebind("gerenteBanco", gc);
            System.out.println("Servidor gerenteBanco pronto.");
        }
        catch (Exception e) {
            System.out.println("BANCO RMI: ocorreu uma exceção - :"+
                e.getMessage());
                e.printStackTrace();
        }
    }

}
```

O Cliente RMI (1)

```
public class ClienteBanco {
    public static void main(String args[]){

        // Verifica a contagem de argumentos
        if( args.length < 2 ){
            System.err.println("Uso:");
            System.err.println("java ClienteBanco <servidor> "+
                               "<nome da conta> [saldo inicial]");
            System.exit(1);
        }

        // Cria e instala o gerente de segurança
        System.setSecurityManager( new RMISecurityManager());
        try {
            // Obtem uma instância para o objeto remoto
            System.out.println("Clinte RMI: pesquisando nome - gerenteBanco");
            String url = new String("rmi://"+args[0]+"/gerenteBanco");
            GerenteConta gc = (GerenteConta) Naming.lookup(url);

            // Define o saldo da conta
            float saldoInicial = 0.0f;
            if( args.length == 3    ) {
                Float F = Float.valueOf(args[2]);
                saldoInicial = F.floatValue();
            }

            // Obtem uma conta (nova ou existe)
            Conta conta = gc.abrir(args[1],saldoInicial);
```



O Cliente RMI (2)



```
            // Obtem uma conta (nova ou existe)
            Conta conta = gc.abrir(args[1],saldoInicial);

            // Realiza alguma operações na implementação do objeto conta
            /*System.out.println("Clinte RMI: saldo = " + conta.getSaldo());
            System.out.println("Clinte RMI: sacando R$ 50,00...");
            conta.saque(50.0f);*/
            System.out.println("Clinte RMI: saldo = " + conta.getSaldo());
            System.out.println("Clinte RMI: depositando R$ 100,00...");
            conta.deposito(100.0f);
            System.out.println("Clinte RMI: saldo = " + conta.getSaldo());
            System.out.println("Clinte RMI: depositando R$ 25,00...");
            conta.deposito(25.0f);
            System.out.println("Clinte RMI: saldo = " + conta.getSaldo());
            System.out.println("Clinte RMI: sacando R$ 50,00...");
            conta.saque(50.0f);
            System.out.println("Clinte RMI: saldo = " + conta.getSaldo());
        }
        catch (Exception e) {
            System.out.println("Cliente RMI: ocorreu uma exceção - :"+
                e.getMessage());
                e.printStackTrace();
        }
        System.exit(1);
    }
}
```