



17 DE NOVEMBRO DE 2016


UNIVALI – CTTMAR – Curso de Ciência da Computação

Disciplina: Programação 2

Prof. Elieser Ademir de Jesus


Pesquisa sobre RMI

Autores: Vinícius Machado, Elmis, Lucas Baragatti e Samuel Favarin





SUMÁRIO

- Introdução ao RMI
 - Definição de RMI
 - Exemplo
 - Conclusão
 - Referências
- 

INTRODUÇÃO

Muitas aplicações necessitam se comunicar com outras. Frequentemente os programadores necessitam implementar um canal de comunicação, para que uma aplicação possa fornecer métodos ou variáveis para a outra. Portanto, existe a necessidade de utilizar o mecanismo de programação RMI (*Remote Method Invocation*), que permite a chamada (invocação) de métodos em diferentes máquinas ou em uma única máquina, facilitando a computação distribuída.

DEFINIÇÃO

RMI é um mecanismo de programação para permitir a invocação (chamada) de métodos que “moram” em diferentes máquinas virtuais Java, sejam elas em LAN, localhost ou em diferentes hosts. Em ambos os casos, o método pode ser executado em um endereço diferente do processo de chamada, ou seja, existe frequentemente um cliente e um servidor. Em Java, RMI é um mecanismo de chamada de procedimento remoto orientada a objetos.

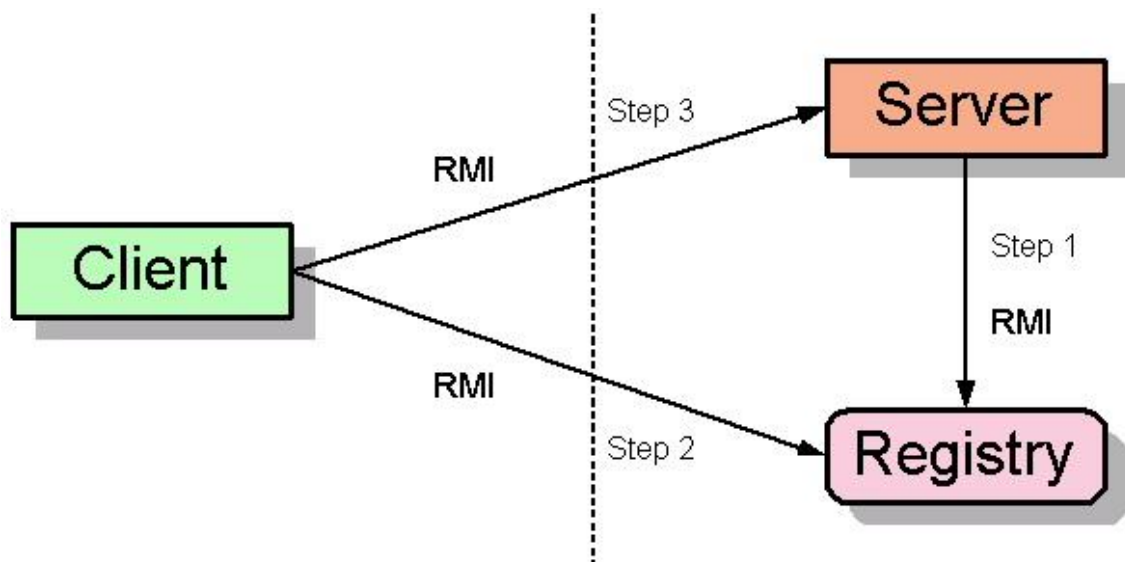


Figura 1 - Ilustração da comunicação Cliente x Server

EXEMPLO EM APLICAÇÃO

Usando RMI, pode-se criar um algoritmo de criptografia, na qual o cliente deseja criptografar uma mensagem. Essa mensagem é enviada para o servidor, que lê a mensagem e retorna-a criptografada ao cliente. O mesmo pode ocorrer quando o cliente deseja traduzir uma mensagem criptografada. O cliente envia a mensagem criptografada e o servidor retorna sua “tradução”.

Quando se desenvolve uma aplicação em Java RMI, alguns elementos básicos devem ser criados, tais como:

- Uma interface que disponibilize os métodos a serem invocados no servidor.
- Uma classe que fique localizada na JVM (Java Virtual Machine) do servidor e que implemente os métodos definidos na interface.
- Classes que implementem o protocolo de comunicação (Skel e Stub)³ e que sejam responsáveis por fazer com que a chamada de um método no cliente seja passada ao servidor de maneira transparente, assim como fazer com que o servidor responda de maneira conveniente a essa chamada, passando de volta ao cliente o valor de retorno.
- Um programa cliente que invoque os métodos remotos do servidor.
- Um serviço de nomes (rmiregistry) responsável por informar ao cliente onde está o servidor e que relacione corretamente a implementação deste ao stub do cliente.

DEFININDO A INTERFACE REMOTA

A interface remota descreve os métodos remotos que o cliente utilizará para interagir com o objeto servidor remoto por RMI.

Para criar uma interface remota, é definido uma interface que estende a interface **Remote** (pacote **java.rmi**). Como mostrado na figura a seguir:

```

6 public interface Criptografia extends java.rmi.Remote {
7     public String criptografar( String a) throws java.rmi.RemoteException;
8     public String descriptografar( String a) throws java.rmi.RemoteException;
9
10 }

```

Figura 2 - Interface Criptografia

A seguir na figura 3, a classe CriptografiaImpl estende a classe remota (java.rmi.server.UnicastRemoteObject) e implementa a classe Criptografia. Existem dois métodos que implementam a interface definida que possuem as seguintes funcionalidades demonstradas nas imagens abaixo:

```

1
2 public class CriptografiaImpl extends java.rmi.server.UnicastRemoteObject
3     implements Criptografia {
4
5     public CriptografiaImpl() throws java.rmi.RemoteException {
6         super();
7     }
8
9     public String criptografar( String a ) throws java.rmi.RemoteException {
10
11         char charArray[];      char charAux=' ';
12         String output = "Texto: " + a;
13         int tam = a.length(); charArray = new char[tam];
14         output += "\n\nCriptografado: ";
15
16         //Obtem a cadeia de caracteres e a coloca em um vetor de char.
17         a.getChars( 0, tam , charArray, 0);
18
19         for (int count =0; count < charArray.length; count++)
20         {
21             if (charArray[count]==' ') charAux=' ';
22             if (charArray[count]=='A' || charArray[count]=='a') charAux='f';
23             if (charArray[count]=='B' || charArray[count]=='b') charAux='j';
24             if (charArray[count]=='C' || charArray[count]=='c') charAux='h';
25             if (charArray[count]=='D' || charArray[count]=='d') charAux='r';
26             if (charArray[count]=='E' || charArray[count]=='e') charAux='p';
27             if (charArray[count]=='F' || charArray[count]=='f') charAux='l';
28             if (charArray[count]=='G' || charArray[count]=='g') charAux='k';
29             if (charArray[count]=='H' || charArray[count]=='h') charAux='z';
30             if (charArray[count]=='I' || charArray[count]=='i') charAux='s';
31             if (charArray[count]=='J' || charArray[count]=='j') charAux='o';
32             if (charArray[count]=='K' || charArray[count]=='k') charAux='u';
33             if (charArray[count]=='L' || charArray[count]=='l') charAux='i';
34             if (charArray[count]=='M' || charArray[count]=='m') charAux='b';
35             if (charArray[count]=='N' || charArray[count]=='n') charAux='v';
36             if (charArray[count]=='O' || charArray[count]=='o') charAux='m';
37             if (charArray[count]=='P' || charArray[count]=='p') charAux='n';
38             if (charArray[count]=='Q' || charArray[count]=='q') charAux='c';
39             if (charArray[count]=='R' || charArray[count]=='r') charAux='x';
40             if (charArray[count]=='S' || charArray[count]=='s') charAux='a';
41             if (charArray[count]=='T' || charArray[count]=='t') charAux='d';

```

Figura 3 - Classe CriptografiaImpl

O primeiro método está escrito a partir da linha nove, onde inicia a declaração de variáveis para que o servidor criptografe uma entrada de string enviada pelo cliente, e a retorne em texto cifrado. Esse método usa a criptografia simétrica com cifra de substituição de caractere, que substitui cada caractere por outro.

```
42         if (charArray[count]=='U' || charArray[count]=='u') charAux='g';
43         if (charArray[count]=='V' || charArray[count]=='v') charAux='q';
44         if (charArray[count]=='X' || charArray[count]=='x') charAux='e';
45         if (charArray[count]=='Z' || charArray[count]=='z') charAux='t';
46
47         charArray[count]= charAux;
48         output+= charArray[count];
49     } //fim for
50     return output;
51 }
52
53 public String descriptografar( String a ) throws java.rmi.RemoteException {
54
55     char charAux = ' ';
56     String output = "Texto Criptografado: " + a;
57     int tam = a.length(); charArray = new char[tam];
58     output += "\n\nTexto Legivel: ";
59
60     //Obtem a cadeia de caracteres e a coloca em um vetor de char.
61     a.getChars( 0, tam , charArray, 0);
62
63     for (int count =0; count < charArray.length; count++)
64     {
65         if (charArray[count]==' ') charAux=' ';
66         if (charArray[count]=='A' || charArray[count]=='a') charAux='s';
67         if (charArray[count]=='B' || charArray[count]=='b') charAux='m';
68         if (charArray[count]=='C' || charArray[count]=='c') charAux='q';
69         if (charArray[count]=='D' || charArray[count]=='d') charAux='t';
70         if (charArray[count]=='E' || charArray[count]=='e') charAux='x';
71         if (charArray[count]=='F' || charArray[count]=='f') charAux='a';
72         if (charArray[count]=='G' || charArray[count]=='g') charAux='u';
73         if (charArray[count]=='H' || charArray[count]=='h') charAux='c';
74         if (charArray[count]=='I' || charArray[count]=='i') charAux='l';
75         if (charArray[count]=='J' || charArray[count]=='j') charAux='b';
76         if (charArray[count]=='K' || charArray[count]=='k') charAux='g';
77         if (charArray[count]=='L' || charArray[count]=='l') charAux='f';
78         if (charArray[count]=='M' || charArray[count]=='m') charAux='o';
79         if (charArray[count]=='N' || charArray[count]=='n') charAux='p';
80         if (charArray[count]=='O' || charArray[count]=='o') charAux='j';
81         if (charArray[count]=='P' || charArray[count]=='p') charAux='e';
82         if (charArray[count]=='Q' || charArray[count]=='q') charAux='v';
83         if (charArray[count]=='R' || charArray[count]=='r') charAux='d';
84         if (charArray[count]=='S' || charArray[count]=='s') charAux='i';
85         if (charArray[count]=='T' || charArray[count]=='t') charAux='z';
86         if (charArray[count]=='U' || charArray[count]=='u') charAux='k';
87         if (charArray[count]=='V' || charArray[count]=='v') charAux='n';
88         if (charArray[count]=='X' || charArray[count]=='x') charAux='r';
89         if (charArray[count]=='Z' || charArray[count]=='z') charAux='h';
90
91         charArray[count]= charAux;
92         output+= charArray[count];
93     }
94     return output;
95 } // fim public String descriptografar
96 } // fim class CriptografiaImpl
```

Figura 4 - Método descriptografar

A linha 53 da figura 4 acima inicia a declaração do método usado para que o servidor descriptografe uma entrada de texto do cliente, e a retorna em texto legível.

Tanto no método criptografar quanto no método descriptografar, o vetor de “chars” é percorrido por meio de um “loop” até que se encontre (Com o auxílio de “IF’s”) a letra desejada para então substituí-la.

Na figura 5 abaixo, está a classe do servidor para “ouvir” as solicitações dos clientes.

```
6 import java.rmi.Naming;
7
8 public class ServidorCriptografia {
9
10     public ServidorCriptografia() {
11         try {
12             Criptografia obj = new CriptografiaImpl();
13             Naming.rebind("//localhost/criptoService", obj);
14         }
15         catch( Exception e) {
16             System.out.println("Erro: " + e);
17         }
18     }
19
20     public static void main(String[] args) {
21         new ServidorCriptografia();
22     }
23 } // fim class ServidorCriptografia
```

Figura 5 - Classe ServidorCriptografia

A linha 13 é a mais importante, pois será onde os clientes iram tentar conexão por meio do nome do objeto servidor (*//host:porta/nomeDoObjetoRemoto*). No exemplo acima é utilizado a conexão ***localhost/criptoService***.

Logo abaixo está representada na figura 6 a classe do cliente com um “try” e inúmeros “catchs” para garantir a conexão com o servidor e execução do programa sem falhas.


```

1  /**
2   * @author Edison da Costa do Nascimento
3   *      Ighor Amaral Souza
4   *      Rodrigo Barroso Gadelha
5   */
6  import javax.swing.*;
7  import java.rmi.Naming;
8  import java.rmi.RemoteException;
9  import java.net.MalformedURLException;
10 import java.rmi.NotBoundException;
11
12 public class ClienteCriptografia {
13
14     public static void main(String[] args) {
15
16         String A, resp = "";
17         //Faz uma Pergunta Sim=0 ou Não=1
18         int opcao = JOptionPane.showConfirmDialog(null, "Sim = Criptografia" +
19             "\nNão = Descriptografia", "Escolha uma Opção", 0);
20         try {
21             Criptografia cripto = (Criptografia) Naming.lookup("//localhost/" +
22                 "criptoService");
23             if (opcao == 0) {
24                 A = JOptionPane.showInputDialog("Entre com o TEXTO. Sem Acento");
25                 resp = cripto.criptografar(A);
26             }
27             else {
28                 A = JOptionPane.showInputDialog("Entre com o TEXTO. Sem Acento");
29                 resp = cripto.descriptografar(A);
30             } //fim else
31         } // fim try
32         catch ( MalformedURLException murle) {
33             System.out.println();
34             System.out.println("MalformedURLException");
35             System.out.println( murle );
36         }
37         catch ( RemoteException re) {
38             System.out.println();
39             System.out.println("RemoteException");
40             System.out.println( re );
41         }
42         catch ( NotBoundException nbe ) {
43             System.out.println();
44             System.out.println("NotBoundException");
45             System.out.println( nbe );
46         }
47         catch ( java.lang.ArithmeticException ae ) {
48             System.out.println();
49             System.out.println("java.lang.ArithmeticException");
50             System.out.println( ae );
51         }
52         catch ( java.lang.StringIndexOutOfBoundsException str ) {
53             System.out.println();
54             System.out.println("java.lang.StringIndexOutOfBoundsException");
55             System.out.println( str );
56         }
57         catch ( java.lang.ArrayIndexOutOfBoundsException arr ) {
58             System.out.println();
59             System.out.println("java.lang.ArrayIndexOutOfBoundsException");
60             System.out.println( arr );
61         } //fim catch
62
63         JTextArea outputArea = new JTextArea();
64         outputArea.setText( resp );
65
66         JOptionPane.showMessageDialog( null, outputArea, "Texto Criptografado"
67             ,JOptionPane.INFORMATION_MESSAGE);
68     } // fim static void main
69 } // fim class
70

```

Figura 6 - Classe do cliente

Quando a classe `ClienteCriptografia` é executada, é feita uma pergunta ao usuário para saber qual serviço será realizado (criptografia ou descriptografia). O cliente, portanto escolhe, entra com o texto e a classe faz uma chamada de método remoto específico para o servidor.

Em seguida, a classe que implementa a interface (**`CriptografiaImpl`**) deve ser compilada utilizando o *compilador **`rmic`*** (um dos utilitários fornecidos com o J2SDK) para produzir uma *classe stub* (sempre será retornado o mesmo valor). Um objeto da classe *stub* permite que o cliente invoque os métodos remotos do objeto servidor. O objeto *stub* recebe cada chamada de método remoto e o passa para o sistema Java RMI, o qual realiza as funções de rede que permitem que o cliente se conecte ao servidor e interaja com o objeto servidor remoto. A linha de comando deve ser executada no diretório em que estão as classes, produzidas pelo compilador `javac`.

Logo abaixo na figura 7 estão presentes as imagens da execução do programa. Inicialmente o usuário deve escolher entre **Sim** (*Criptografia*) e **Não** (*Descriptografia*) no Frame invocado. Após a escolha, o usuário deve entrar com o texto sem acento, clicar em “OK” e esperar o retorno do texto criptografado ou descriptografado:

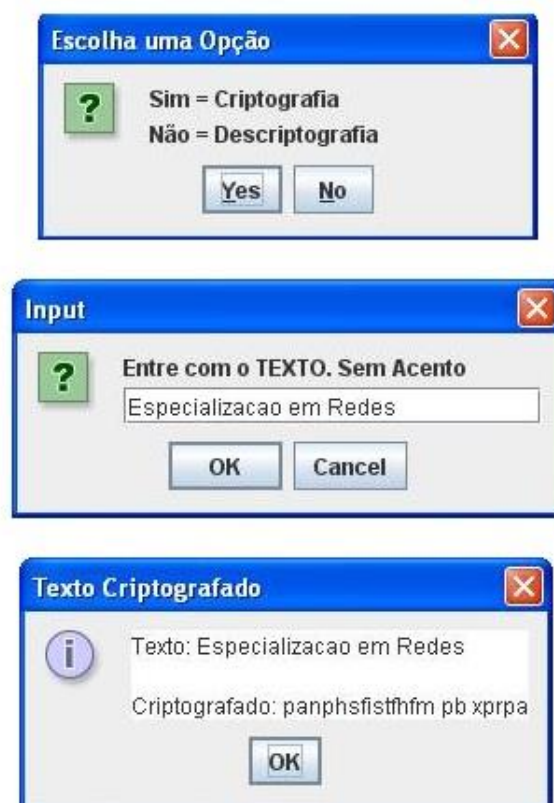


Figura 7 - Execução do programa

REFERÊNCIAS

Edílson da Costa do Nascimento. **“Exemplo prático do uso de RMI em sistemas distribuídos: Serviço de Criptografia”**.<http://www.linhadecodigo.com.br/artigo/2831/exemplo-pratico-do-uso-de-rmi-em-sistemas-distribuidos-servico-de-criptografia.aspx>. Acessado no dia 17/11/2016 as 10:03.

Welington. (2011). “Teste Unitário: O que é um Stub?”.
<https://wsilva81.wordpress.com/2011/09/03/teste-unitario-o-que-um-stub/>.
Acessado no dia 18/11/2016 as 20:20.

Coulouris, G.; Dollimore, J.; Kindberg, T (2007). “Sistemas Distribuídos - conceito e projeto”. Editora Bookman