Invocação de Métodos Remotos Aplicação: Chat multi-thread com java-RMI

Lucas Alves Da Costa.

[1]*LucasAlves16.LA@gmail.com*

[2]Instituto federal de educação, ciência e tecnologia-Goiás
(IFG)
Matricula: *20151070130183.*

Resumo. *Este relatório mostra o funcionamento da aplicação:* ***Chat com Java RMI****. O uso da Thread foi justamente para capacitar que os clientes possam receber as mensagens que foram enviadas para uma lista contida no servidor em quanto enviam mensagens para o mesmo, ou seja, um bate papo onde os clientes enviam a mensagens para o servidor, e o mesmo as repassam a través do canal estabelecido pela thread, este projeto implementado se limita ao uso interface gráfica (java.swing) pois a mesma não foi implementada até o momento, e também há o risco de algumas exceções remotas serem lançadas, isso impedira o funcionamento correto de um programa cliente em particular. essas exceções não foram tratadas de modo que o cliente continue a execução normalmente, ao invés disso o programa que lançar uma exceção deverá ser reexecutado. não foi testado com exatidão o limite máximo de clientes conectados ao servidor, acredito que isso vareie de maquina para maquina.*

1. Java RMI

O que é RMI? Java RMI é um mecanismo para permitir a invocação de métodos que residem em diferentes máquinas virtuais Java (JVM). O JVM pode estar em diferentes máquinas ou podem estar na mesma máquina. Em ambos os casos, o método pode ser executado em um endereço diferente do processo de chamada. Java RMI é um mecanismo de chamada de procedimento remoto orientada a objetos.

Uma aplicação RMI é frequentemente composto por dois programas diferentes, um servidor e um cliente. O servidor cria objetos remotos e faz referências a esses objetos disponíveis. Em seguida, ele é válido para clientes invocarem seus métodos sobre os objetos.

O cliente executa referências remotas aos objetos remotos no servidor e invoca métodos nesses objetos remotos.

O modelo de RMI fornece uma aplicação de objetos distribuídos para o programador. Ele é um mecanismo de comunicação entre o servidor e o cliente para se comunicarem e transmitirem informações entre si. A aplicação de objetos distribuídos tem de prover as seguintes propriedades:

Localização de objetos remotos: O sistema tem de obter referências a objetos remotos. Isto pode ser feito de duas maneiras. Ou, usando as instalações de nomeação do RMI, o registro RMI, ou passando e retornando objetos remotos. Comunicação com objetos remotos: O desenvolvedor não tem de lidar com a comunicação entre os objetos

remotos desde que este é tratado pelo sistema RMI. Carregar os bytecodes de classe dos objetos que são transferidos como argumentos ou valores. Todos os mecanismos para carregar o código de um objeto e transmitissão dos dados são fornecidos pela RMI system.40

Implementação: A aplicação foi feita visando os conceitos de RMI, com implementação em Java, veja abaixo a implementação da classe servidor.

```
1  package jrmi.chat;
2  import java.rmi.Naming;
3  import java.rmi.registry.LocateRegistry;
4  import java.rmi.registry.Registry;
5
6  public class Servidor {
7  public Servidor(){
8
9      try {
10 Registry registry = LocateRegistry.createRegistry(1098);
11 InterfaceChat server = new ServidorChatImpl();
12 Naming.rebind("rmi://localhost:1098/InterfaceChat",server);
13
14 } catch (Exception e){
15 System.out.println("Exception:"+e.getMessage());
16 }
17
18
19 }
20
21 public static void main (String args[]){
22  new Servidor();
23  System.out.println("Servidor online.");
24 }
25
26 }
```

2. Servidor

Esta classe é responsável por criar registrar e exportar o objeto remoto criado, note que nas linha 10; é criado um registro na porta padrão 1098 usada pela RMI neste caso.

Na linha 11 é criado um objeto do tipo da classe onde está a implementação dos métodos de troca de mensagens, o mesmo exportado para o stub local através do método descrito na linha 12, com o método **Naoming.rebind** é possível criar o nome do serviço que será usado pelos clientes. O objeto **server**, passa a ser construído, com o construtor da classe, e cadastrado no sistema operacional, através do método rebind(String str, Remote r) da classe Naming. A classe Naming fornece um mecanismo para obter referencias a objetos remotos baseados na sintaxe da URL. Onde uma URL para objetos remotos é usualmente especificada pelo Host, porta e nome: rmi://host:porta/nome. Veja abaixo a implementação dos métodos do servidor para administrar as mensagens.

```
1
```

```
2  package jrmi.chat;
3  import java.rmi.RemoteException;
4  import java.rmi.server.UnicastRemoteObject;
5  import java.util.ArrayList;
6
7  public class ServidorChatImpl extends java.rmi.server.
       UnicastRemoteObject implements InterfaceChat {
8  ArrayList<String> mensagens;
9  int nMensagens;
10 public ServidorChatImpl() throws RemoteException {
11 super();
12 this.mensagens = new ArrayList<String>();
13 }
14
15 public void enviarMensagem(String mensagem) throws
       RemoteException{
16 mensagens.add(mensagem);
17 }
18
19 public ArrayList<String> lerMensagem() throws RemoteException{
20 return mensagens;
21
22 }
23
24 }
```

A Logica utilizada aqui é simples, implementamos os métodos da interface que será utilizada pelo cliente e pelo servidor na invocação dos métodos. Na linha 8 é criado uma lista de Strings que servirá para armazenarmos as mensagens enviadas pelos clientes conectados, na linha 11 temos o construtor da classe que irá instanciar um novo objeto do tipo da lista que foi definida anteriormente. Na linha 16 temos o método **enviarMensagem()** que é utilizada pelos clientes para mandar mensagens para o servidor, a mensagem recebida do cliente é adicionada ao topo da lista, isso pode ser visto na linha 17. O método **lerMensagem()** descrito na linha 20 é utilizado pelos clientes para receber as mensagens enviadas para o servidor, seu funcionamento é simples, o método é invocado e o servidor devolve a lista contendo as mensagens recebidas. Veremos agora a interface mencionada anteriormente, veja abaixo sua implementação.

```
1
2  package jrmi.chat;
3
4  import java.rmi.Remote;
5  import java.rmi.RemoteException;
6  import java.util.ArrayList;
7
8  public interface InterfaceChat extends Remote {
9  public void enviarMensagem(String mensagem) throws
       RemoteException;
10 public ArrayList<String> lerMensagem() throws RemoteException;
11 }
```

Todas as interfaces RMI, devem ser derivadas da classe java.rmi.Remote e devem ter o tratamento de suas exceções realizadas através da *java.rmi.RemoteException*, para que as exceções sejam repassadas remotamente. Note que os métodos descritos nessa interface java são a assinatura dos mesmos métodos implementados no servidor, isso é obrigatório por todas as classes que implementarem a interface em questão, essa interface servirá para que os clientes conectados possam utiliza-los de maneira transparente, ou seja, apenas se preocuparão em invocar esses métodos sem saber sua implementação. Vendo de maneira conceitual: Toda aplicação Java é construída a partir de interfaces e classes, logo, da mesma forma é uma aplicação distribuída construída com Java RMI. A interface declara métodos. As classes implementam os métodos declarados na interface e possivelmente conterá métodos adicionais. Numa aplicação distribuída, algumas implementações podem permanecer em alguma Java VM mas não em outras. Objetos com métodos que podem ser invocados entre Java VM são chamados de Objetos Remotos. Para se tornar um Objeto Remoto, a classe deve implementar uma interface remota:

3. Cliente

A implementação da Classe Cliente que irá se beneficiar dos métodos remotos é um pouco mais extensa, mas o funcionamento é semelhante ao do servidor, note que essa classe também implementa os métodos da interface descrita, mesmo sendo uma classe mais extensa a assinatura dos métodos são os mesmos. Nesta classe o cliente poderá enviar uma mensagem para o servidor e receber as mensagens dos outros que também foram enviadas para o servidor. Analisemos o código abaixo:

```
1
2  package jrmi.chat;
3
4  import java.rmi.Naming;
5  import javax.swing.*;
6  import java.util.Scanner;
7  import java.util.ArrayList;
8  import java.rmi.RemoteException;
9  import java.rmi.registry.LocateRegistry;
10 import java.rmi.registry.Registry;
11
12 public class ClienteChat {
13
14 public static void main( String args[] ) {
15
16
17 try {
18 final InterfaceChat chat = (InterfaceChat) Naming.lookup( "rmi
      ://localhost:1098/InterfaceChat");
19 final int controle =1;
20 String nome ="";
21 String msg = "";
22
23 Scanner scanner = new Scanner(System.in);
24
25
```

```
26
27 while (nome == null || nome.equals("")) {
28     nome=(JOptionPane.showInputDialog("Qual o seu nome?"));
29     chat.enviarMensagem("\t\t\t"+nome+" entrou no grupo...\n");
30    }
31
32 System.out.printf("Bem Vindo ao bate-papo da UOL: "+nome+"\
      nDigite sua mensagem;\n");
33 Thread thread = new Thread(new Runnable() {
34 int cont = chat.lerMensagem().size();
35
36 @Override
37 public void run() {
38 try {
39
40 while(controle ==1){
41 if (chat.lerMensagem().size() > cont){
42 System.out.println(chat.lerMensagem().get(chat.lerMensagem().
      size()-1));
43 cont++;
44 }
45 }
46 } catch (RemoteException e) {
47   System.out.printf("Ocorreu um erro, reinicie o programa!\n");
48   System.out.println("Exe   o: "+e.getMessage());
49   System.exit(0);
50
51 }
52 }
53 });
54 thread.start();
55 int sair =0;
56 while(sair == 0){
57        msg = scanner.nextLine();
58        System.out.printf("-->");
59
60 if(msg.equals("1") || msg.toString()=="1"){
61     sair = 1;
62     chat.enviarMensagem("\t\t\t"+nome+" Saiu do grupo!\n");
63     System.exit(0);
64 }
65
66 chat.enviarMensagem("\t\t\t<-- "+nome+":("+msg+")\n");
67 msg ="";
68 }
69
70 }
71 catch( Exception e ) {
```

```
72      System.out.println("Exe   o:2 "+e.getMessage()+"Exe   o 2"
            );
73  }
74  }
75  }
```

Na linha 18 temos a instanciação do objeto que receberá o serviço (objeto remoto) que utilizaremos nesta classe, o método **Naoming.lookup()** é configurado com o IP do servidor e o serviço a ser utilizado pelo cliente, esse método irá procurar no seu stub local por esse objeto remoto, ao encontra-lo poderemos usa-lo em nossa classe. Da linha 27 até a linha 30 temos um laço que força o cliente a inserir um nome de usuário para ser usado na aplicação, este nome será usado como identificação nas mensagens enviadas ao servidor. Note que na linha 29 é enviada uma mensagem para o servidor através do método **enviarMensagem()** informando que um usuário entrou no grupo, essa mensagem será repercutida a todos os clientes conectados. Da linha 33 até a linha 57 está a implementação da Thread usada pelo cliente em questão para buscar as mensagens enviadas para o servidor, utilizamos para isso o método remoto **lerMensagem()** que recebe como argumento outro método (linha 42) que garantirá que iremos receber apenas as ultimas mensagens enviadas para o servidor, sem isso receberíamos a conversa todas as mensagens da lista, inclusive as que já recebemos anteriormente, tornando enviável o dialogo com os outros clientes conectados. Assim que um cliente se conecta ao servidor já é possível receber e enviar mensagens para o mesmo, essa mensagem será repassada a todos os clientes conectados a esse servidor. A parte responsável por ler e enviar nossa mensagem se inicia na linha 56 onde podemos notar um laço que nos permite escrever e enviar mensagens para o servidor enquanto a opção de saída não for digitada pelo cliente, essa opção se refere a linha 60 onde verificamos se o cliente digitou a opção de saída, ou seja, o numero 1. Note na linha 62 que se o cliente optar em sair será enviada uma mensagem ao servidor informando que o mesmo saiu do grupo Se a opção (1) não foi digitada, assim que a tecla 'enter' for pressionada a mensagem digitada pelo cliente é enviada para o servidor através do método **enviarMensagem()** onde passamos como argumento; o nome do cliente em questão, a mensagem digitada por ele e algumas tabulações para organizarmos a conversa no console, podemos observar esse processo na linha 66. Essa mensagem será inserida na lista de mensagens do servidor, e será recebida pelos clientes conectados, inclusive este!.

4. Considerações finais

Foi mostrado neste relatório o funcionamento de uma aplicação baseada em Java RMI onde Clientes podem se comunicar através de um servidor, a pesar de ser uma aplicação simples, os principais conceitos da RMI foram mostrados através da implementação em java. Com essa implementação torna-se fácil utilizar um RMI já que não é necessário conhecer as abstrações da implementação ou até mesmo a linguagem utilizada pelo outro processo, é necessário apenas conhecer as assinaturas dos métodos propostos pela interface e também (enviar/receber) um objeto devidamente preparado. Tendo em vista a implementação anterior podemos concluir que o sistema, tanto cliente quanto servidor pode evoluir sua implementação desde que as assinaturas dos métodos continuem as mesmas. Para utilizar este serviço é necessário criar um RMI Registry em cada maquina hospedada especificamente na pasta do servidor, mas o intuito aqui é mostrar o funcionamento desta aplicação. Todo código foi compilado e testado antes de ser explicado.