RMI - Remote Method Invocation
Tutorial
Sistemas Distribuídos

Alifi Cleiton, Haltielles César, Natan Morais, Tiago Melo

19 de março de 2017

Professor: Rafael Frinhani

1 Objetivo

O objetivo deste tutorial é demonstrar passo-a-passo a preparação e desenvolvimento necessário para que seja possível a utilização da interface de comunicação RMI (Remote Method Invocation) em um programa Java. Ao final deste tutorial, será possível que seu sistema cliente invoque algumas operações matemáticas que se localiza no código do servidor remoto.
O passo-a-passo será dado utilizando a IDE Netbeans para a criação do sistema e execução da mesma, e o sistema operacional utilizado será o Ubuntu Linux, portanto algumas modificações podem surgir como necessárias para adequar esse tutorial à outros sistemas.

2 Preparação Ambiente de Desenvolvimento

Para este tutorial, utilizaremos o ambiente de desenvolvimento Netbeans. Portanto, se o leitor não possuir ainda o sistema, poderá obtê-lo no link:
Netbeans Download.
Depois de baixá-lo, será apenas necessário instalar da forma que julgar correta, ou apenas seguir os passos do instalador. Após instalado, tenha certeza de que o Netbeans tem os plugins desejados para o desenvolvimento Java. Plugins que o ambiente deseja que você instale, os outros disponíveis e os já instalados podem ser acessados em *Tools -> Plugins*.
Ao garantir que ambiente de desenvolvimento é capaz de criar aplicações Java, também tem-se a certeza de que é possível utilizar o RMI nas mesmas, pois suas classes e funções são nativas da linguagem.

3 Desenvolvimento Java

Para o início do desenvolvimento, é necessário criar um projeto. Então é necessário que se entre em *File -> Novo Projeto...* ou *Ctrl+Shift+N*, onde aparecerá uma tela com escolhas para o tipo de projeto que será desenvolvido.
Nesta tela, escolha a categoria *Java*, e nos projetos escolha *Java Application*. Clique em *Next*. Na próxima tela, escolha um nome para seu projeto e um caminho onde ele será armazenado em seu computador. O checkbox onde aparece a opção de criação automática da classe principal deve ser desabilitada. Clique em *Finish*.
Temos agora então um projeto em branco. É necessário criar agora então as classes que conterão todo o código necessário para que tenhamos uma comunicação utilizando RMI. Para criar uma classe, clique com o botão direito acima do projeto que acabou de ser criado e selecione a opção *Novo -> Java Class...*. Para o caso em que este tutorial lida, a única opção que é necessário alterar é o nome da classe, e depois clicar em **Finish**. Para fins de fácil entendimento, e facilidade no continuamento do desenvolvimento, cria-se todas as classes necessárias de uma vez só, estas são: **Client**, **Server**, **Calculator**, **ICalculator**, **ClientTest**, **ServerTest**.
O código das classes serão indicadas abaixo, junto com as devidas explicações necessárias:

class Client

```
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * Cliente.
 * Responsavel por acessar um objeto remotamente por
 * meio de um IP e um nome.
 *
 * @param <T> Tipo do objeto a ser acessado.
 */
public class Client<T extends Remote>
{
    private static final int PORT = 1098;
    private final T remote;

    /**
     * Cria uma instancia da classe Client.
     *
     * @param ip    Endereco IP do local onde se quer
     * acessar o objeto.
```

```
     * @param name Nome do objeto a ser acessado.
     */
    public Client(String ip, String name)
            throws RemoteException, NotBoundException,
            MalformedURLException
    {
        //Acessa um objeto remotamente atraves do ip do
        //local onde esta o objeto e seu nome.
        remote = (T)Naming.lookup("rmi://" + ip + ":" +
                                        PORT + "/" + name);
    }

    /**
     * Obtem o objeto remoto.
     */
    public T getRemoteObj()
    {
        return remote;
    }
}
```

Esta classe Client contém apenas duas coisas, mas de extrema importância para o funcionamento da comunição do Client para com o Server. Ele contém a porta de acesso em que o cliente vai enviar a requisição para o servidor, indicado pela variável *PORT*.
Outra coisa que há neste método é a chamada na função *Naming.lookup()*, onde é feita a requisição do cliente para o servidor. Ele contém o IP e porta do servidor, além do nome do serviço que está sendo buscado.

class Server

```
import java.rmi.Remote;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

/**
 * Servidor.
 * Contem o objeto que sera acessado remotamente.
 */
public class Server
{
    private static final int PORT = 1098;

    /**
     * Cria uma instancia da classe Server.
     *
```

```
     * @param remote Objeto que sera acessado remotamente.
     * @param ip     Endereco IP.
     * @param name   Nome do objeto.
     */
    public Server(Remote remote, String ip, String name)
            throws IllegalAccessException,
            InstantiationException
    {
        init(remote, ip, name);
    }

    private void init(Remote remote, String ip,
                      String name)
    {
        try
        {
            //Cria um registro que aceita pedidos pela
            //porta especificada.
            Registry registry =
                      LocateRegistry.createRegistry(PORT);
            //Caminho com o IP, porta e nome.
            String uri = "rmi://" + ip + ":" + PORT + "/"
                                                  + name;
            System.out.println(uri);
            //Vincula o caminho com um objeto que sera
            //acessado remotamente.
            Naming.rebind(uri, remote);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

A classe Server, ao contrário do que a classe anterior é responsável, tem como funcionalidade a ligação do Servidor com qualquer cliente que queira acessá-lo. Ele abre um registro, indicando a porta pelo qual um cliente pode acessar o servidor. Depois ele indica à quem buscar algum serviço o IP de acesso ao serviço e o nome do mesmo.

class Calculator

```
import java.rmi.server.UnicastRemoteObject;

public class Calculator extends UnicastRemoteObject
```

```
                        implements ICalculator
{
    protected Calculator()
            throws RemoteException {}

    public double soma(double a, double b)
            throws RemoteException
    {
        System.out.println(a + "," + b);
        return a + b;
    }
}
```

A classe Calculator serve apenas para o servidor, é onde está as funções ou serviços que o servidor contém. Diferentemente da classe ICalculator que será vista abaixo, aqui existe todas as implementações dos serviços do servidor.

class ICalculator

```
import java.rmi.RemoteException;

public interface ICalculator extends Remote
{
    double soma(double a, double b)
            throws RemoteException;
}
```

Já a interface ICalculator fornece um "contrato"dos serviços que o servidor pode gerar. Ela é utilizada pelo cliente para que saiba a forma (modelo) com que os serviços são chamados e o que elas retornam. Não há qualquer tipo de implementação de serviço nesta interface.

class ClientTest

```
import java.rmi.NotBoundException;
import java.rmi.RemoteException;

public class ClientTest
{
    public static void main(String[] args)
    {
        try
        {
            Client<ICalculator> client = new Client<>
                        ("IP_ADDRESS", "SERVICE_NAME");
            System.out.println(client.getRemoteObj()
                                .soma(VALOR_1, VALOR_2));
```

```
        }
        catch(RemoteException e)
        {
            e.printStackTrace();
        }
        catch(NotBoundException e)
        {
            e.printStackTrace();
        }
        catch(MalformedURLException e)
        {
            e.printStackTrace();
        }
    }
}
```

Esta classe é apenas uma classe executável para o cliente, onde ele usa as interfaces contidas na classe anterior e a função de conexão que foi apresentado na classe Client.

class ServerTest

```
import java.rmi.RemoteException;

public class ServerTest
{
    public static void main(String[] args)
    {
        try
        {
            Calculator calc = new Calculator();
            Server server = new Server(calc, "IP_ADDRESS",
                                       "SERVICE_NAME");
        }
        catch(IllegalAccessException e)
        {
            e.printStackTrace();
        }
        catch(InstantiationException e)
        {
            e.printStackTrace();
        }
        catch(RemoteException e)
        {
            e.printStackTrace();
        }
```

```
    }
}
```

O ServerTest, assim como a classe ClientTest, é apenas uma classe executável, desta vez para o servidor. É nela que existe a inicialização da classe que contém os serviços do servidor e a chamada de função que estabelece a conexão que está presente na classe Server.

Obs.: *IP_ADDRESS*: É o endereço de IP onde o serviço está sendo executado. Na classe ServerTest ele deve apontar um IP do próprio Server disponível para acesso. Na classe ClientTest esse endereço deve apontar o endereço que o Server disponibilizou para ser usado. No fim das contas, ambos acabam tendo o mesmo valor.

SERVICE_NAME: Deve ser o mesmo nas classes ClientTest e ServerTest, é o nome pelo qual o serviço pode ser chamado.

VALOR_1 e *VALOR_2*: Valores passados como referência para o serviço chamado, neste caso, dois valores para uma soma.

Após a criação de todos essas classes, que também estão disponíveis em um repositório do Github, é necessário executá-los, para isso, temos dois casos:

- **Conexão localhost:** Clique com o botão direito sobre a classe ServerTest e selecione a opção *Executar Arquivo*. Neste momento, um terminal da IDE se abrirá e não se encerrará, significa que o Servidor está disponível para funcionar e receber requisições. Então, clique com o botão direito em ClientTest e repita o procedimento. Será possível verificar um novo terminal surgindo. No terminal do Servidor teremos elementos que retratam a requisição que acabou de ser recebida, já no terminal do Cliente teremos o resultado da operação (soma, neste caso) que foi executada no Servidor.

- **Conexão remota:** Para uma conexão remota, é necessário que haja ao menos dois computadores diferentes que contenham as classes. Desta vez um computador será responsável por executar apenas a classe ServerTest, para que então o outro computador execute o ClientTest e ambos receberem os respectivos resultados como citado no item anterior.

Para garantir que os testes funcionem corretamente, é necessário ter todos os códigos apresentados e sem modificações que atrapalhem seu funcionamento. Além disso, é necessário que o computador que está sendo executado o servidor, seja numa conexão remota ou até mesmo local (se está utilizando IP da máquina e não 127.0.0.1), desabilite o seu firewall para que possa ser acessado. No windows isso pode ser feito pelo painel de controle, entrando nas opções de Firewall, no linux é mais simples e basta utilizar a seguinte linha de comando:

```
$ sudo ufw disable
```

E por último, é necessário que no Linux, ainda no computador que serve como servidor, este altere o arquivo */etc/hosts*, tirando o IP 127.0.1.1 que aparece

como IP do perfil que está sendo utilizado e troque pelo IP que o servidor está disponibilizando como acessível para que clientes tenham seu serviço.
Feito todos esses passos, você tem a garantia de estar rodando um programa que utiliza RMI como comunicação entre diferentes sistemas!