

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
PUC Minas Poços de Caldas
Ciência da computação

Samuel Tunes
Maria Amélia Doná Aguilar

Trabalho 02 - Java RMI

Poços de Caldas
2023

Lista de ilustrações

Figura 1 – Funcionamento do 'rmiregistry'(RODOLFO,)	4
Figura 2 – Código do Servidor do Java RMI	5
Figura 3 – Resultado da execução do Server.java no terminal	6
Figura 4 – Código do Cliente do Java RMI	6
Figura 5 – Resultado da execução do Client.java no terminal	7
Figura 6 – Código da Interface do Java RMI	7

1 Introdução

Foi apresentado na aula de Laboratório de Redes de Computadores, ministrada pelo professor Harison, onde a proposta foi a de pesquisar sobre Java RMI(Remote Method Invocation) e por sua vez mostrar um exemplo de código do funcionamento em prática do Java RMI. Que por sua vez é uma tecnologia da plataforma Java que permite a comunicação e invocação de métodos entre objetos distribuídos em diferentes máquinas virtuais Java. Em resumo RMI (Remote Method Invocation) é uma interface de programação que permite a execução chamadas remotas desenvolvidas em Java

2 Desenvolvimento

O RMI facilita a comunicação entre aplicativos Java em um ambiente distribuído, permitindo que objetos Java chamem métodos em objetos remotos localizados em máquinas diferentes, como se estivessem chamando métodos em objetos locais. O RMI cuida de toda a complexidade subjacente de comunicação em rede, serialização de objetos e transporte de dados entre as JVMs.

Usando o RMI, é possível criar aplicativos distribuídos em Java, onde os objetos podem invocar métodos uns nos outros mesmo que estejam em diferentes máquinas. Isso permite construir sistemas distribuídos complexos, como sistemas de computação em grade, sistemas cliente-servidor escaláveis, aplicativos distribuídos em larga escala e muito mais.

O RMI é baseado no modelo de objeto remoto, onde os objetos remotos são registrados em um registro RMI e podem ser acessados por outros objetos através de suas interfaces remotas. O RMI usa a serialização para transferir objetos entre as JVMs, permitindo que os objetos sejam passados como parâmetros e retornados como resultados das invocações de método remoto.

O Java RMI é uma tecnologia poderosa para desenvolvimento de sistemas distribuídos em Java, fornecendo uma abordagem simples e orientada a objetos para a comunicação entre objetos distribuídos, para melhor entendimento de como funciona o Java RMI, é fundamental entender o fluxo de comunicação.

2.1 Como funciona o Java RMI

Para criar um aplicativo usando Java RMI, é necessário definir as interfaces remotas que descrevem os métodos que podem ser invocados remotamente, implementar as classes que implementam essas interfaces e configurar o registro RMI para registrar e localizar objetos remotos.

O Java RMI lida com muitos detalhes de comunicação em rede, serialização de objetos, gerenciamento de referências e tratamento de exceções automaticamente, permitindo que você se concentre na lógica.

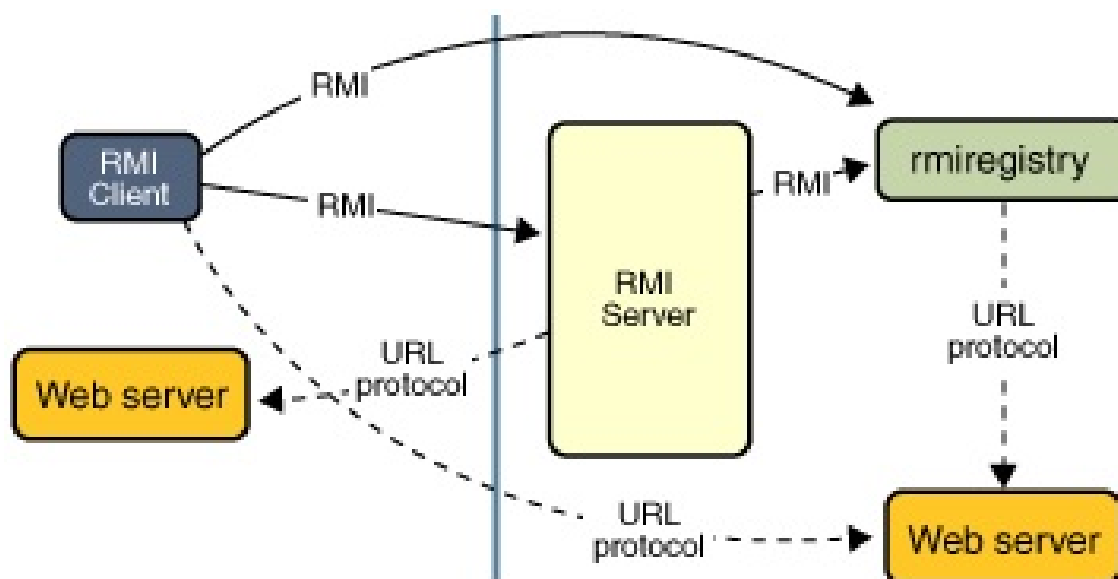
A serialização no RMI permite que objetos Java sejam transmitidos pela rede e utilizados como se estivessem no mesmo sistema. Ela desempenha um papel fundamental na transparência do RMI, permitindo que os métodos remotos sejam invocados em objetos remotos de forma transparente para o desenvolvedor.

Outro conceito importante é o de Stub e Skeleton que são usados para facilitar a comunicação entre objetos remotos. Stub é a representação local do objeto remoto no cliente, enquanto o Skeleton é o intermediário entre o Stub e o objeto remoto no servidor. Eles desempenham um papel essencial no RMI, simplificando a comunicação e permitindo

que os objetos remotos sejam invocados de forma transparente pelos clientes.

Em uma aplicação RMI (Remote Method Invocation), existem dois componentes principais: o cliente e o servidor. O servidor é responsável por criar um objeto remoto, associá-lo a um nome e registrá-lo em uma porta específica, aguardando as chamadas dos clientes. Por sua vez, o cliente remoto se refere aos métodos do objeto remoto por meio do nome registrado previamente. O RMI proporciona um mecanismo de comunicação eficiente entre cliente e servidor. Para isso, utiliza o “rmi-registry”, que permite que o aplicativo distribuído obtenha as referências necessárias aos objetos remotos.

Figura 1 – Funcionamento do 'rmiregistry'(RODOLFO,)



2.2 Códigos

Os códigos foram baseados na explicação do site (DEV MEDIA,), onde nele apresenta um simple exemplo de comunicação usando RMI, porém o exemplo apresentado aqui será ainda mais simples, que por sua vez apenas apresentará o servidor, o cliente e a interface.

2.2.1 Servidor

O código em `Server.js` a seguir implementa um servidor RMI (Remote Method Invocation) que disponibiliza um objeto remoto para comunicação com clientes. Onde será explicado de forma breve como funciona cada parte do código:

Figura 2 – Código do Servidor do Java RMI



```
1  /*Equipe: Maria Amélia
2     Samuel*/
3  import java.rmi.RemoteException;
4  import java.rmi.registry.LocateRegistry;
5  import java.rmi.registry.Registry;
6  import java.rmi.server.UnicastRemoteObject;
7
8  public class Server implements Interface {
9
10     public Server() {
11         super();
12     }
13
14     public String teste() throws RemoteException {
15         return "Comunicação simples até o client!";
16     }
17
18     public static void main(String[] args) {
19         try {
20
21             Server server = new Server();
22             Interface stub = (Interface) UnicastRemoteObject.exportObject(server, 0);
23             Registry registry = LocateRegistry.createRegistry(1099);
24             registry.rebind("Boanoite", stub);
25
26             System.out.println("Servidor RMI pronto para receber requisições.");
27         } catch (Exception e) {
28             System.err.println("Erro no servidor: " + e.toString());
29         }
30     }
31 }
```

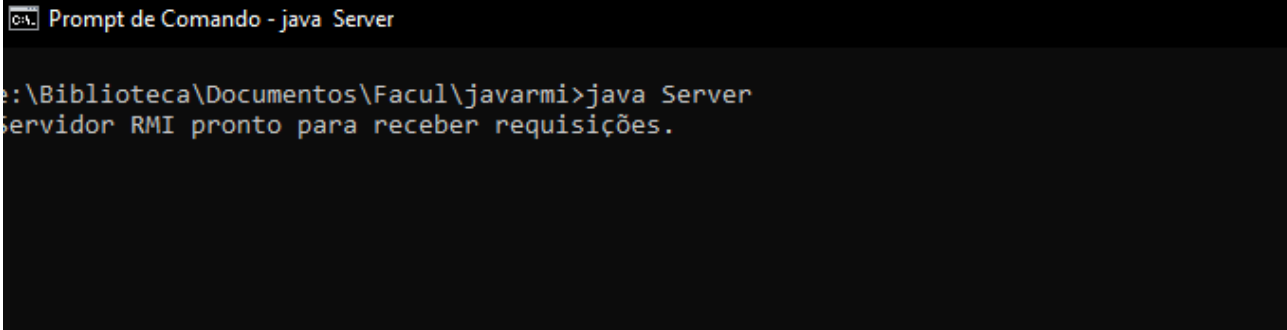
Esse código cria um servidor RMI que disponibiliza um objeto remoto chamado “Boanoite” para os clientes. O objeto remoto implementa a interface `Interface` e possui um método `teste()`, que retorna uma resposta simples para os clientes. O servidor aguarda as requisições dos clientes na porta 1099.

De forma mais detalhada a classe `Server` implementa a interface `Interface`, que deve definir os métodos remotos disponíveis para os clientes. Logo em seguida o construtor da classe `Server` é chamado, e a classe `super()` é invocada para chamar o construtor da classe pai (nesse caso, a classe `Object`).

O método `teste()` é implementado e lança uma exceção `RemoteException`. Esse método será invocado pelos clientes para obter uma resposta do servidor.

O método `main()` é o ponto de entrada do programa. Nele, temos o seguinte fluxo: Um objeto `Server` é criado. O método `UnicastRemoteObject.exportObject()` é chamado para exportar o objeto remoto. Isso faz com que o objeto seja disponibilizado para chamadas remotas. Um registro RMI é criado na porta 1099 utilizando `LocateRegistry.createRegistry()`. O registro é responsável por armazenar as referências aos objetos remotos. O objeto remoto é registrado no registro RMI usando `registry.rebind()`. O nome “Boanoite” é associado ao objeto remoto, permitindo que os clientes o acessem por esse nome.

O resultado da execução em um terminal pode ser visto a seguir:

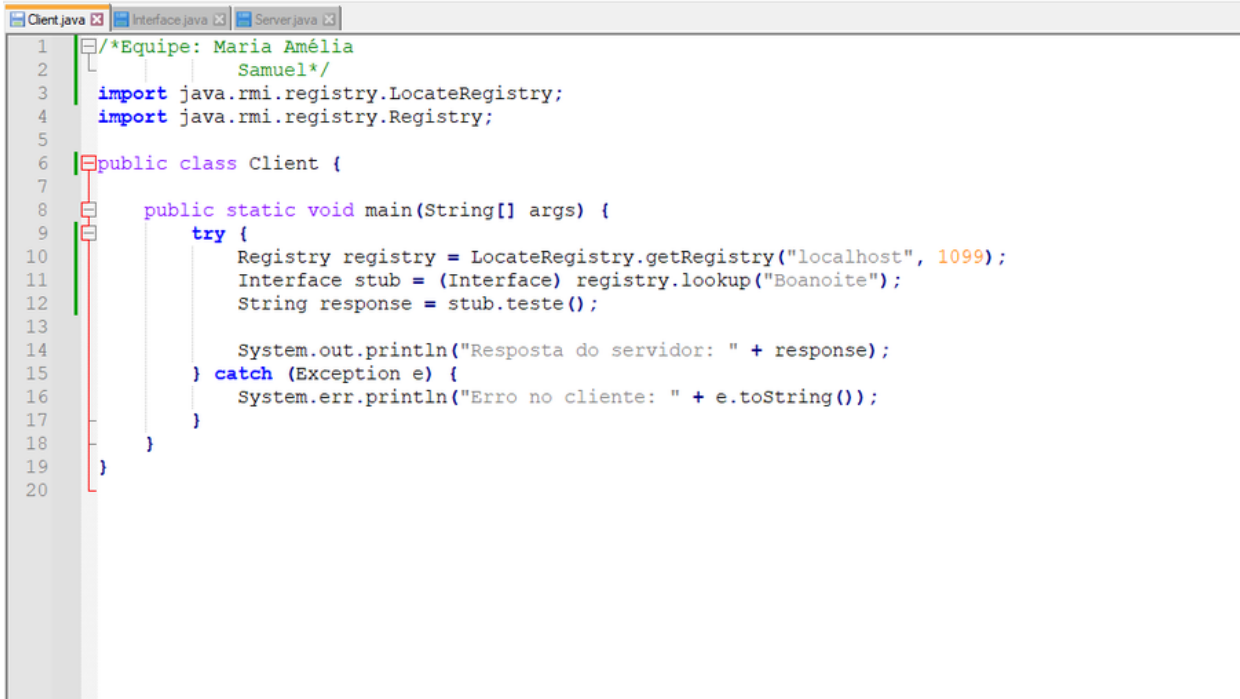
Figura 3 – Resultado da execução do Server.java no terminal

```
Prompt de Comando - java Server

e:\Biblioteca\Documentos\Facul\javarmi>java Server
servidor RMI pronto para receber requisições.
```

2.2.2 Cliente

O código a seguir implementa um cliente RMI (Remote Method Invocation) que se conecta a um servidor RMI e invoca um método remoto. Onde será explicado de forma breve como funciona cada parte do código:

Figura 4 – Código do Cliente do Java RMI

```
Client.java | Interface.java | Server.java
1  /*Equipe: Maria Amélia
2      Samuel*/
3  import java.rmi.registry LocateRegistry;
4  import java.rmi.registry Registry;
5
6  public class Client {
7
8      public static void main(String[] args) {
9          try {
10             Registry registry = LocateRegistry.getRegistry("localhost", 1099);
11             Interface stub = (Interface) registry.lookup("Boanoite");
12             String response = stub.teste();
13
14             System.out.println("Resposta do servidor: " + response);
15         } catch (Exception e) {
16             System.err.println("Erro no cliente: " + e.toString());
17         }
18     }
19 }
20
```

esse código implementa um cliente RMI que se conecta a um servidor RMI, recupera a referência a um objeto remoto chamado “Boanoite” e invoca o método remoto ‘teste()’. A resposta do servidor é exibida no console do cliente.

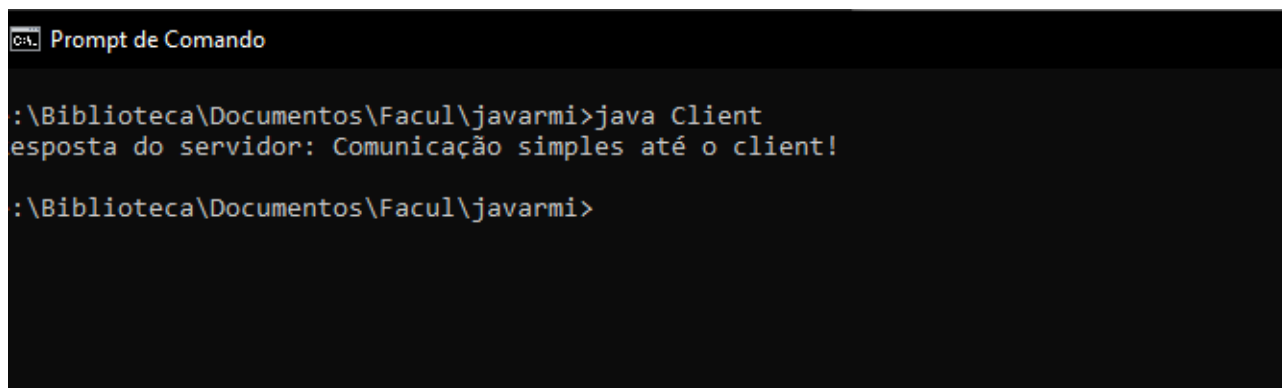
O método main() é o ponto de entrada do programa. Nele, temos o seguinte fluxo:

É obtida uma referência ao registro RMI utilizando 'LocateRegistry.getRegistry()'. Nesse caso, o registro está sendo acessado localmente na porta '1099'.

É obtida a referência ao objeto remoto registrado no registro RMI usando 'registry.lookup()'. Nesse caso, o nome "Boanoite" é utilizado para recuperar a referência ao objeto remoto.

O método remoto 'teste()' é invocado no objeto remoto através da referência stub, e o resultado é armazenado na variável response.

Figura 5 – Resultado da execução do Client.java no terminal



```
Prompt de Comando

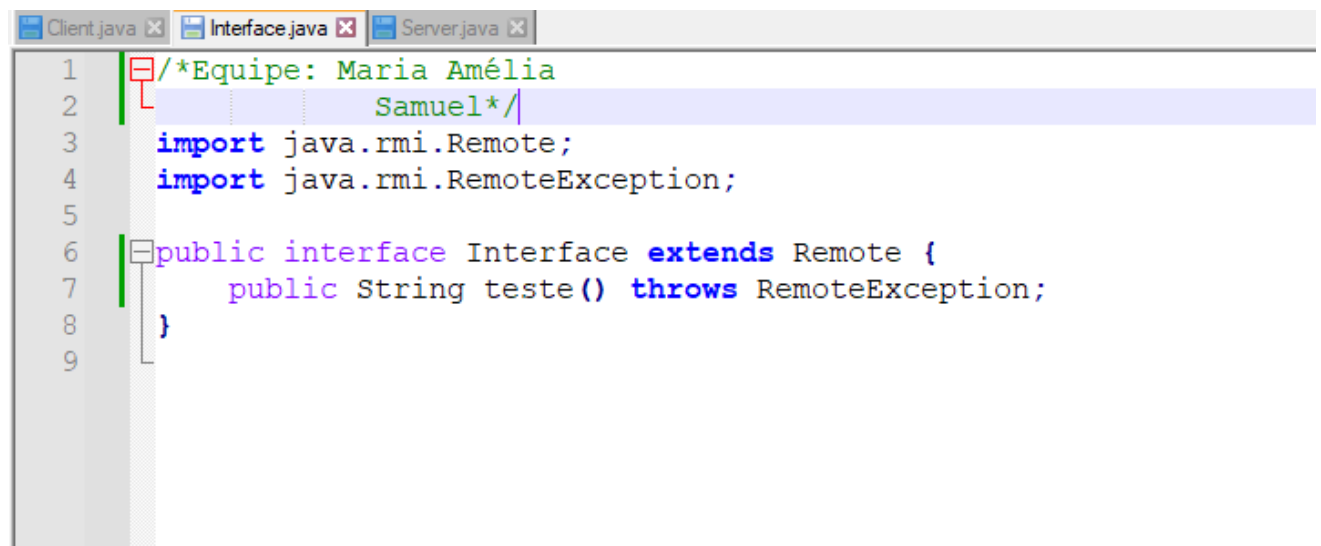
:\\Biblioteca\\Documentos\\Facul\\javarmi>java Client
esposta do servidor: Comunicação simples até o client!

:\\Biblioteca\\Documentos\\Facul\\javarmi>
```

2.2.3 Interface

Este código implementa uma interface chamada Interface que estende a interface Remote do RMI (Remote Method Invocation) Onde será explicado de forma breve como funciona cada parte do código:

Figura 6 – Código da Interface do Java RMI



```
Client.java x Interface.java x Server.java x
1  /*Equipe: Maria Amélia
2     Samuel*/
3  import java.rmi.Remote;
4  import java.rmi.RemoteException;
5
6  public interface Interface extends Remote {
7      public String teste() throws RemoteException;
8  }
9
```


Esse código define uma interface RMI chamada `Interface` que contém um único método `'teste()'`. Essa interface será implementada tanto pelo servidor RMI quanto pelo cliente RMI para permitir a comunicação e a invocação remota dos métodos definidos na interface.

A interface `'Interface'` é declarada. Ela estende a interface `'Remote'`, que é a marcação do RMI para indicar que essa interface contém métodos que podem ser invocados remotamente.

Dentro da interface, temos a declaração de um único método: `'teste()'`. Esse método não recebe argumentos e retorna uma string. A declaração `'throws RemoteException'` indica que esse método pode lançar uma exceção do tipo `RemoteException`, que é uma exceção padrão do RMI que indica problemas de comunicação remota.

Referências

- DEVMEDIA. **Uma introdução ao RMI em Java**. Disponível em: <https://www.devmedia.com.br/uma-introducao-ao-rmi-em-java/28681>. Acesso em: 01/06/2023.
- RODOLFO, I. O. e L. **Java RMI**. Disponível em: <https://deinfo.uepg.br/~alunoso/2017/RMI/#contato>. Acesso em: 01/06/2023.