1. O que é JAVA RMI

RMI (Remote Method Invocation) é uma API (Application Programming Interface) que possibilita, como o próprio nome diz, invocar métodos de objetos remotos, ou seja, permite que objetos que estejam em outras máquinas sejam invocados pela sua máquina. É bastante semelhante ao RPC (Remote Procedure Call), porém usa uma abordagem orientada a objetos.

2. Java RMI

O sistema RMI é composto por 3 camadas de abstração.

- Stubs e Skeletons

  Bastante parecido com o modelo RPC, o RMI usa Stubs, porém apenas no lado do Cliente e apenas Skeletons no lado do Servidor. Quando o Cliente invoca um método de um objeto o Stub toma o controle, abrindo um soquete de rede com o Servidor, transmitindo os parâmetros do método e esperando pelo resultado. O Skeleton, por sua vez, fica ouvindo na porta estabelecida esperando por chamadas a métodos, esse realiza a invocação do método passando os parâmetros recebidos, e quando finalizado o Skeleton retorna para o Stub pelo soquete o resultado da invocação. O Stub, então retorna para a invocação o resultado recebido.

  A função dessa função é passar os dados para a camada de referências remotas, através de marshal streams, e receber os dados recebidos pela camada de referências, através de unmarshalling.

  Marshalling é o processo de transformar um objeto em um representação de dados com o objetivo de armazená-lo ou transportá-lo. O termo marshalling é usado para se referir a objetos remotos, quando são os objetos locais é usado o termo serialização, que é transformar estruturas de dados em sequências de bits.

  Ao transformar um objeto em um marshal stream do RMI, o dados deve ser capaz de ser reconstruído no lado do Servidor, dependo do hardware essa tarefa pode ser um pouco difícil, já que diferentes máquinas possuem diferentes codificações de memória, little-endian e big-endian. Felizmente para o desenvolvedor o RMI lida sozinho com esse problema.

- Referências Remotas

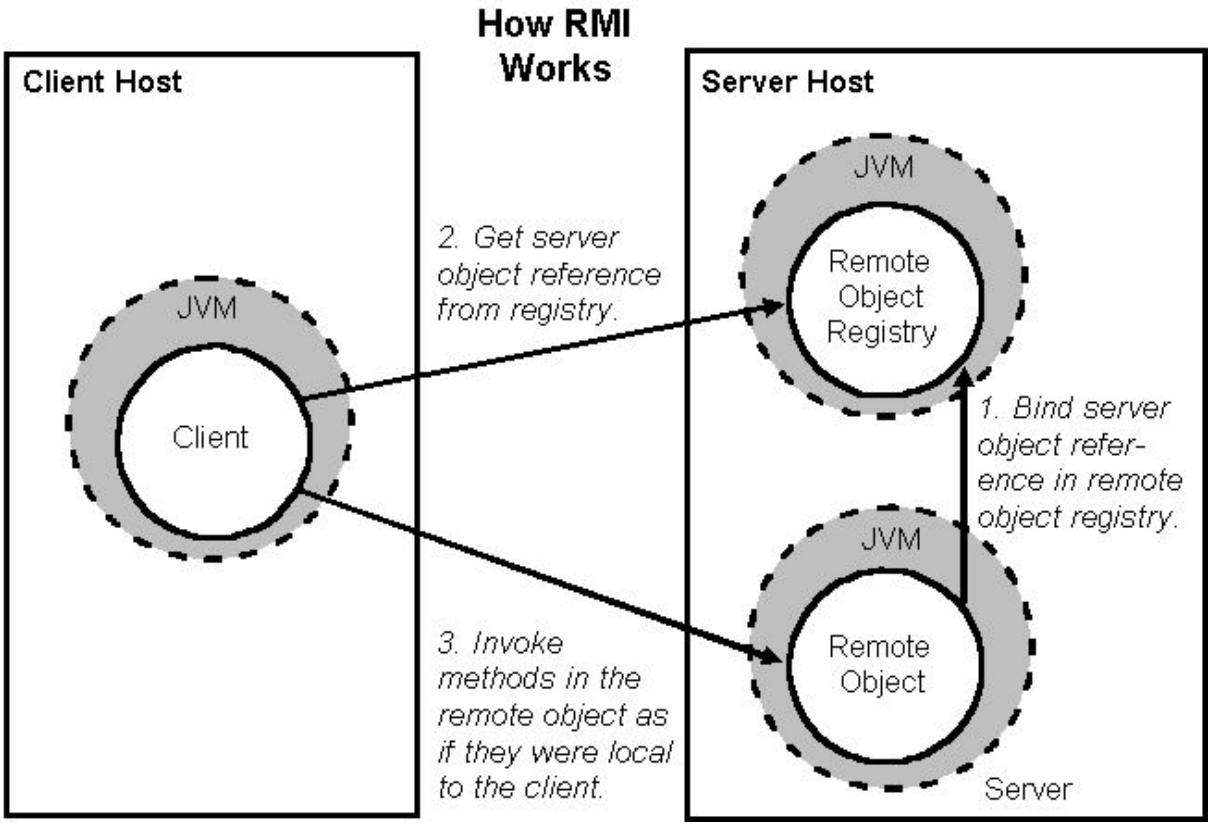
  RRL (Remote Reference Layer), como o nome já diz, é a camada de referências remotas, ou seja, é a camada do RMI responsável por gerenciar e armazenar as referências dos objetos remotos. Quando é invocado um método de um objeto é invocado o sistema tem que saber qual objeto remoto possui esse método, pois

pode haver mais de um objeto remoto, e como acessá-lo, é para isso que essa camada é usada.

O RMI usa o objeto LocateRegistry para obter uma referência a um objeto remoto. Ao criar um registro é dado um nome que irá associar esse nome com o objeto remoto, fazendo uma ligação(binding) com o nome e o objeto.

O registro de objetos remotos é rodado no servidor, quando um Cliente deseja fazer uma invocação primeiro ele precisa pegar as informações dos registros dos servidores para depois ele então fazer uma chamada para um servidor que possui esse objeto registrado.

Como pode ser visto na imagem abaixo, os registros ficam apenas nos Servidores.



Além disso essa camada também tem que ser responsável por implementar diferentes protocolos de invocação, já que cada objeto pode possuir um diferente protocolo:

Unicast é um dos possíveis protocolos. É feita uma invocação ponto-a-ponto, diferente do próximo.

Multicast também é um dos possíveis protocolos. É feito a transmissão para diversos computadores da mesma rede.

Suporte para um estratégia de replicação específica, e estratégias de reconexão também são alguns dos protocolos usados. Nem sempre apenas um protocolo é usado para transmitir, podendo combiná-los.

- Camada de Transporte

  Essa camada é de maneira geral responsável por seis atividades. São elas: configurar as conexões com espaços de endereçamentos remotos, administrar as conexões, monitorar a queda de uma conexão, ouvir novas chamadas, manter a tabela de objetos remotos e configurar uma conexão para realizar novas chamadas.

  A implementação desta camada é bastante influenciado por Modula-3, uma ferramenta desenvolvida na década de 80, porém nunca foi adotada de forma considerável. A camada de transporte do RMI pode ser subdividido em quatro camadas sendo elas:

  1. Endpoint: é um espaço de endereçamento, que mapeia o transporte. Dado um endpoint pode ser obtido o transporte.
  2. Transporte: é a camada de abstração que gerencia os canais, além de ser responsável por aceitar as chamadas, ajeitar as conexões para a chamada, e enviar as informações para as camadas mais altas do RMI.
  3. Canal: é o canal entre dois endereços, sendo responsável por gerenciar conexões entre o Cliente e o Servidor.
  4. Conexão: é a transferência de dados (input e output).

3. Protocolo de Transmissão

O Java RMI usa dois protocolo para transmissão, o Java Object Serialization para transformar o dado em um marshal stream e o HTTP para enviar e receber os dados transmitidos quando ele é necessário.

O formato do dado transmitido é representado por uma stream, podendo ser In, de input, e Out, de output. Para cada In há um Out correspondente, e vice e versa.

O Output normalmente é composto por um header e seguido por mensagens, há casos em que a invocação de algum método por está dentro do protocolo HTTP. Segue um exemplo de Out:

```
Out:
    Header Messages
    HttpMessage

Header:
    0x4a 0x52 0x4d 0x49 Version Protocol

Version:
    0x00 0x01

Protocol:
    StreamProtocol
    SingleOpProtocol
    MultiplexProtocol

StreamProtocol:
    0x4b

SingleOpProtocol:
    0x4c

MultiplexProtocol:
    0x4d

Messages:
    Message
    Messages Message
```

O Input diferente do Output pode ter três tipos de mensagens:

1. ReturnData: é o resultado “normal” de um chamada RMI.
2. HttpReturn: é o resultado de uma invocação dentro de um protocolo HTTP.
3. PingAck: é a comfirmação de uma mensagem de ping.

Abaixo segue o exemplo de um In:

```
In:
    ProtocolAck Returns
    ProtocolNotSupported
    HttpReturn

ProtocolAck:
    0x4e

ProtocolNotSupported:
    0x4f

Returns:
    Return
    Returns Return

Return:
    ReturnData
    PingAck

ReturnData:
    0x51 ReturnValue

PingAck:
    0x53
```

4. Aplicação

O exemplo de aplicação usando Java RMI será uma calculadora simples, que pode fazer quatro operações (soma, subtração, divisão e multiplicação). Esse programa foi feito em um Ubuntu, e é necessário ter instalado JDK.

- Calculadora.java (Inteface Calculadora)

```java
import java.rmi.Remote;

public interface Calculadora extends Remote{
    public double calc(double a,double b, char c) throws RemoteException;
}
```

- CalculadoraClient.java (Classe CalculadoraClient)

Assim como o nome diz é a execução do Cliente, responsável pela invocações dos métodos dos objetos remotos (nesse caso o método calc da Calculadora).

```java
import java.rmi.Naming;
public class CalculadoraClient {
    public static void main(String[] args) {
        try {
            Calculadora c = (Calculadora) Naming.lookup("rmi://192.168.0.15:1099/CalculadoraService");
            Scanner l = new Scanner(System.in);
            double a,b;
            char op;
            int d = 0;
            while(d == 0){
                System.out.println("0-Calculadora\n1-Sair");
                d = l.nextInt();
                if(d == 0){
                System.out.println("Informe os valores a serem calculados e a operação na seguinte forma -> <Valor1> <Operação> <Valor2>:");
                System.out.println("Valor 1");
                a=l.nextDouble();
                System.out.println("Operador");
                op = (char)System.in.read();
                System.out.println("Valor 2");
                b=l.nextDouble();
                System.out.println("Resultado : " + c.calc(a,b,op));
                }
            }
            l.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Na linha 6 é necessário colocar o IP do Servidor, como por exemplo 192.168.0.15, e a porta, que nesse caso é 1099.

- CalculadoraServer.java (Classe CalculadoraServer)

Responsável por fazer os registros de objetos remotos para que possa ser feito a invocação.

```java
import java.rmi.Naming;

public class CalculadoraServer {
    CalculadoraServer() {
        try {
            System.setProperty("java.rmi.server.hostname","192.168.0.15");
            LocateRegistry.createRegistry(1099);
            Calculadora c = new CalculadoraImple();
            Naming.bind("CalculadoraService", (Remote) c);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        new CalculadoraServer();
    }
}

```

No linha 8 é preciso novamento registrar o IP do Servidor, e na linha 9 registar a mesma porta que está no linha 6 da CalculadorClient.

- CalculadoraImple.java (Classe CalculadoraImple)

  Objeto que será invocado.

```
1  import java.rmi.RemoteException;
3
4  public class CalculadoraImple extends UnicastRemoteObject implements Calculadora{
5
6      private static final long serialVersionUID = 1L;
7
8      protected CalculadoraImple() throws RemoteException{
9          super();
10     }
11     public double calc(double a, double b, char op) throws RemoteException{
12         double i = 0;
13         if(op == '+') {
14             i = this.add(a, b);
15         }else if(op == '-') {
16             i = this.sub(a, b);
17         }else if(op == '*') {
18             i = this.mult(a, b);
19         }else if(op == '/') {
20             i = this.div(a, b);
21         }
22         return i;
23     }
24     private double add(double a, double b){
25         System.out.println("A = " + a + " B = " + b);
26         return a+b;
27     }
28     private double sub(double a, double b){
29         System.out.println("A = " + a + " B = " + b);
30         return a-b;
31     }
32     private double mult(double a, double b){
33         System.out.println("A = " + a + " B = " + b);
34         return a*b;
35     }
36     private double div(double a, double b){
37         System.out.println("A = " + a + " B = " + b);
38         return a/b;
39     }
40 }
41
```

- Criação do stub e execução

Depois de Criar a interface e as classes é preciso compilar os arquivos. Isso pode ser feito pela IDE ou por meio do comando:
javac <NomeDoArquivo>
Após todos arquivos compilados é preciso criar o stub, para isso é preciso ir no terminal, entrar no diretório onde os arquivos estão e executar o seguinte comando:
rmic CalculadorImple
Se ocorreu tudo certo, agora só precisa executar o Servidor e depois o Cliente. Pode ocorrer um erro ao tentar gerar o stub devido a versão do Java.

5. Referências

http://www.sce.carleton.ca/netmanage/simulator/rmi/RMIExplanation.htm
https://www.tutorialspoint.com/java_rmi/java_rmi_introduction.htm
http://mrbool.com/rmi-remote-method-invocation-in-java/28575
https://docs.oracle.com/javase/tutorial/rmi/overview.html

https://www.javatpoint.com/RMI#:~:text=The%20RMI%20(Remote%20Method%20Invocation,two%20objects%20stub%20and%20skeleton.
https://docs.oracle.com/javase/9/docs/specs/rmi/protocol.html
http://www.dca.fee.unicamp.br/cursos/PooJava/objdist/javarmi.html
https://web.fe.up.pt/~eol/AIAD/aulas/JINIdocs/rmi1.html
https://docs.oracle.com/javase/8/docs/platform/rmi/spec/rmi-arch2.html
https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html#define%60
https://www.devmedia.com.br/uma-introducao-ao-rmi-em-java/28681
https://www.ibm.com/support/knowledgecenter/en/SSYKE2_7.0.0/com.ibm.java.win.70.doc/diag/understanding/rmi_implementation.html
https://www.ibm.com/support/knowledgecenter/SSYKE2_7.0.0/com.ibm.java.win.70.doc/diag/understanding/rmi.html
https://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/guide/rmi/spec/rmi-arch.doc.html
https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2017_2/rmi/arquitetura.html
https://pdos.csail.mit.edu/archive/6.824-2006/papers/rmi96.pdf