

# RMI JAVA

Guilherme Moura Baccarin

24 de Junho de 2022





# RMI

**Java RMI** é um mecanismo para permitir a invocação de métodos que residem em diferentes máquinas virtuais Java (JVM).

JVM pode estar em diferentes máquinas ou podem estar na mesma máquina

Uma aplicação RMI é frequentemente composta por dois programas diferentes, um servidor e um cliente. O servidor cria objetos remotos e faz referências a esses objetos disponíveis.

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

Ele é um mecanismo de comunicação entre o servidor e o cliente para se comunicarem e transmitirem informações entre si.



# 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.



# Interfaces e Classes

Java RMI é um sistema de linguagem individual

Todas as interfaces e classes para o sistema de RMI são definidos no pacote `java.rmi`

A classe de objeto remoto implementa a interface remota, enquanto as outras classes estendem `RemoteObject`



# A interface remota

Uma interface remota é definida pela extensão da interface Remote que está no pacote java.rmi.

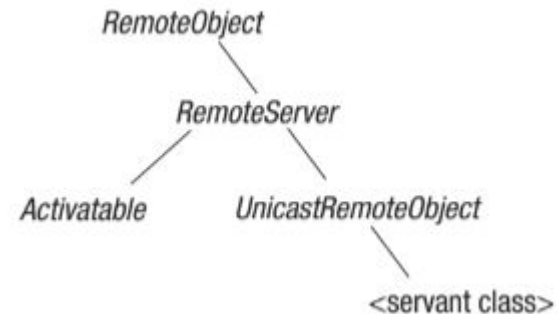
A interface que declara os métodos que os clientes podem invocar a partir de uma máquina virtual remoto é conhecida como interface remota. A interface remota deve satisfazer às seguintes condições:

- Deve estender-se a interface Remote.
- Cada declaração de método na interface remota deve incluir a exceção RemoteException (ou uma de suas superclasses), em sua cláusula lançada.

# A classe RemoteObject

Funções do servidor RMI são fornecidos pela classe RemoteObject e suas subclasses Remote Server, Activatable e UnicastRemoteObject.

- RemoteObject fornece implementações dos métodos toString, equals e hashCode na classe java.lang.Object.
- As classes UnicastRemoteObject e Activatable cria objetos remotos e os exporta, ou seja, essas classes fazem os objetos remotos usados por clientes remotos.





# Implementação de sistema RMI

Para criar o sistema RMI todos os arquivos tem que estar compilados. Em seguida, o outline e stub, que são mecanismos de comunicação padrão com objetos remotos, são criadas com o compilador RMIC

Exemplo:

- Hello.java: A interface remota.
- HelloClient.java: A aplicação cliente no sistema RMI.
- HelloServer.java: O aplicativo de servidor no sistema RMI.
- Quando todos os arquivos são compilados, executando o seguinte comando irá criar o stub e o skeleton: `RMIC HelloServer`

```

1 public class HelloServerDemo extends UnicastRemoteObject
2 implements Hello {
3     public HelloServerDemo() throws RemoteException { super(); }
4
5     public String Hello() {
6         System.out.println(x: "Invocation to Hello was succesful!");
7         return "Hello World from RMI server!";
8     }
9

```

Run | Debug

```

10 public static void main(String args[]) {
11     try {
12         // Creates an object of the HelloServer class.
13         HelloServer obj = new HelloServer();
14         // Bind this object instance to the name "HelloServer".
15         Naming.rebind("Hello", obj);
16         System.out.println(x: "Ligado no registro");
17     }
18     catch (Exception ex) {
19         System.out.println("error: " + ex.getMessage());
20         e.printStackTrace();
21     }
22


```

```

1 public class HelloClient {
2     ...
3     public static void main(String args[]){
4         try {
5             obj = (Hello)Naming.lookup("//kvist.cs.umu.se/Hello");
6             message = obj.Hello();
7             System.out.println(x: "Mensagem no servidor RMI de: message ");
8         }
9         catch (Exception e) {
10             System.out.println("HelloClient exception: " + e.getMessage());
11             e.printStackTrace();
12         }
13     }
14 }

```





```
2  public class CalculatorClient{
    Run | Debug
3  public static void main(String[] args)
4  {
5      try{
6          Calculator c = (Calculator) Naming.lookup("//127.0.0.1:1020/CalculatorService");
7          System.out.println("Adição : "+c.add(20, 15));
8      }
9      catch (Exception e)
10     {
11         System.out.println(e);
12     }
13 }
14 }
```

```
2  public class CalculatorServer{
3      CalculatorServer()
4      {
5          try{
6              Calculator c = new CalculatorImple();
7              Naming.rebind("RMI://127.0.0.1:1020/CalculatorService", c);
8          }
9          catch (Exception e){
10             e.printStackTrace();
11         }
12     }
    Run | Debug
13 public static void main(String[] args)
14 {
15     new CalculatorServer();
16 }
17 }
```



# Execução

- `cd Java\jdk1.6.0_23\bin`
- `javac Calculator.java`
- `javac CalculatorImple.java`
- `javac CalculatorServer.java`
- `javac CalculatorClient.java`
- `rmic CalculatorImple` iniciar registro RMI
- `java CalculatorServer`
- `cd e:\java1\jdk1.6.0_23\bin`
- `java CalculatorClient`

# RMI JAVA

Guilherme Moura Baccarin

24 de Junho de 2022

