# RMI

## Sumario


- 1 Que é RMI
- 2 Funcionamento
- 3 Interfaces, obxectos e métodos remotos
- 4 Pasos para crear unha aplicación RMI
- 5 Carga dinámica de código
- 6 Exemplo
- 7 Referencias


## Que é RMI

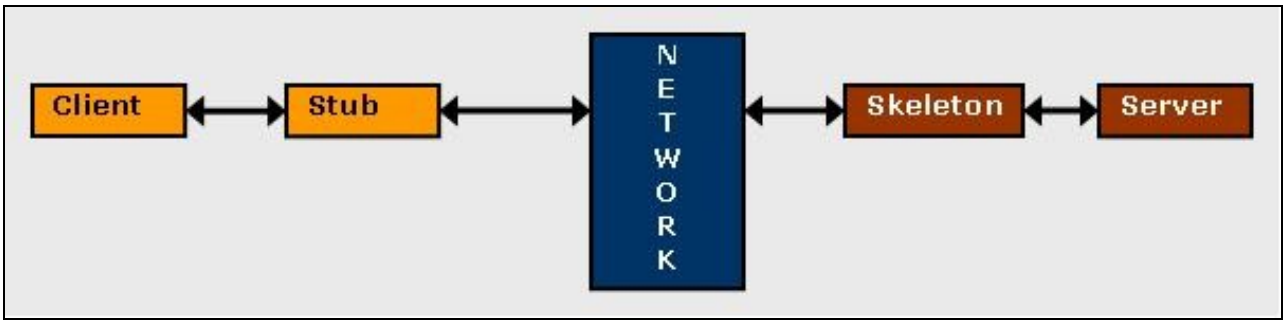
A tecnoloxía de Invocación Remota de Métodos (RMI) de Java permite a un obxecto que se está executando nunha Máquina Virtual Java (JVM) chamar a métodos doutro obxecto que está noutra JVM diferente, posiblemente noutro equipo. Polo tanto, pode entenderse como unha mistura entre a programación distribuída e o paradigma de orientación a obxectos, permitindo construír aplicacións que se poden executar de forma distribuída empregando unha rede TCP/IP.

As aplicacións RMI normalmente comprenden dous programas separados: un servidor e un cliente. Unha aplicación servidor típica crea unha chea de obxectos remotos, fai accesibles unhas referencias a devanditos obxectos remotos, e espera a que os clientes chamen a estes métodos ou obxectos remotos. Unha aplicación cliente típica obtén unha referencia remota de un ou máis obxectos remotos no servidor e chama aos seus métodos. RMI proporciona o mecanismo polo que o cliente e o servidor se comunican e se pasan información.

RMI está soportado na linguaxe Java polo paquete `java.rmi`

## Funcionamento

Grosso modo, o funcionamento de RMI é o seguinte:




- Mediante un compilador especial, o **RMIC**, xéranse dous servidores intermedios (proxies), un da parte do cliente, chamado **stub**, e outro da parte servidor, chamado **skeleton** (dende a versión Java 2 o skeleton non é necesario)
- Cando o cliente invoca un método o stub envía a petición do método xunto cos seus parámetros **serializados** (*marshalling* en inglés) pola rede ao skeleton. Ese paso é posible grazas ao **Object Registry**, que é un servidor de nomes que contén obxectos e os seus nomes. É dicir, os obxectos están rexistrados no **Registry**, mediante o cal se pode acceder a un obxecto remoto empregando o seu nome.
- Despois, o skeleton **deserializa** os parámetros e pásaos ao servidor que, baseándose neles, obtén o resultado que se envía ao cliente polo camiño inverso.

Para acceder ao RMI Registry, e polo tanto aos nomes dos obxectos, úsase a clase estática `Naming` que proporciona o método `lookup()` que o cliente usa para consultar o rexistro. O método acepta un URL que especifiza o nome do equipo e o servizo desexado. O URL ten a seguinte forma:

```
rmi://<nome_equipo>[:<porto_nome_servizo>]/<nome_servizo>
```

O porto só precisa ser especificado se é distinto de 1099.

## Interfaces, obxectos e métodos remotos

Unha aplicación construída utilizando RMI, do mesmo xeito que outras aplicacións Java, está composta por interfaces e clases. Os interfaces definen métodos, mentres que as clases implementan os métodos definidos nos interfaces e, quizais, tamén definen algúns métodos adicionais. Os obxectos que teñen métodos que poden chamarse por distintas máquinas virtuais son os **obxectos remotos**.

Un obxecto convértese en remoto implementando un interface remoto, que teña estas caracterísitcas.

- O interface remoto **herda** do interface `java.rmi.Remote`.
- Cada método do interface declara que lanza unha `java.rmi.RemoteException` ademais de calquera excepción específica da aplicación.

## Pasos para crear unha aplicación RMI

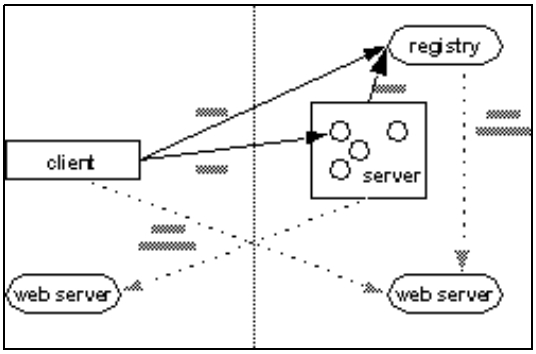
Os pasos que se adoita seguir para crear unha aplicación remota usando RMI son os seguintes:

1. Definir os **interfaces remotos**. Un interface remoto especifica os métodos que se poden chamar remotamente por un cliente. Os clientes programan os interfaces remotos, non a implementación das clases de devanditos interfaces.
2. Implementar os **obxectos remotos**. Os obxectos remotos teñen que implementar un ou varios interfaces remotos. A clase do obxecto remoto pode incluír implementacións doutros interfaces (locais ou remotos) e outros métodos (que só estarán dispoñibles localmente).
3. Implementar os **clientes**. Os clientes que utilizan obxectos remotos poden ser implementados logo de definir os interfaces remotos, incluso despois de que sexan despregados os obxectos remotos .
4. Compilar os ficheiros de código fonte e **xerar o stub e o skeleton**. Este é un proceso de dous pasos. No primeiro paso, utilízase o compilador **javac** para compilar os ficheiros fonte, os cales conteñen as implementacións dos interfaces remotos, as clases do servidor, e do cliente. No segundo paso é utilizar o compilador **rmic** para crear o stub e o skeleton dos obxectos remotos.
5. **Arrincar a aplicación**. Isto inclúe executar o rexistro de obxectos remotos de RMI, o servidor e o cliente.

## Carga dinámica de código

Unha das principais características de RMI é a habilidade de descargar os bytecodes dunha clase dun obxecto se a clase non está definida na máquina virtual do receptor. Os obxectos, anteriormente só dispoñibles nunha soa máquina virtual, agora poden ser transmitidos a outra máquina virtual, posiblemente remota. O comportamento de devanditos obxectos non cambia cando son enviados a outra máquina virtual. Isto permite que os novos obxectos sexan introducidos en máquinas virtuais remotas, e así estender o comportamento dunha aplicación dinamicamente.

A seguinte ilustración mostra unha aplicación RMI distribuída que utiliza o rexistro para obter referencias a obxectos remotos. O servidor chama ao rexistro para asociar un nome cun obxecto remoto. O cliente busca o obxecto remoto polo seu nome no rexistro do servidor e logo chama a un método. Esta ilustración tamén mostra que o sistema RMI utiliza unha servidor Web existente para cargar os bytecodes da clase Java, desde o servidor ao cliente e desde o cliente ao servidor, para os obxectos que necesita.




## Exemplo

Os conceptos expostos poden verse neste exemplo

## Referencias

Para ampliar información recomendo as seguintes seguintes referencias:

- Páxina principal de Sun sobre RMI: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
- Titorial sobre RMI da Sun Developer Network (SDN): http://java.sun.com/developer/onlineTraining/rmi/

- API, diferentes titoriais e ferramentas de desenvolvemento: http://java.sun.com/javase/6/docs/technotes/guides/rmi/index.html