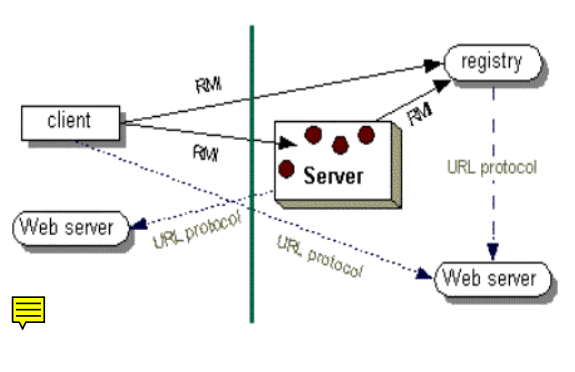
# Funcionamento do RMI

O funcionamento de RMI consiste basicamente em dois programas, um seria o cliente e outro o servidor. O servidor instancia objetos remotos, o referência com um nome, e faz um vinculo ( bind ) dele numa porta, onde este objeto espera por clientes que invoquem seus métodos. Já o cliente referencia remotamente um ou mais métodos de um objeto remoto. RMI fornece os mecanismos para que a comunicação entre cliente e servidor seja possível.

Aplicações distribuídas usando RMI precisam executar as seguintes ações:

* Localizar objetos remotos – uma aplicação pode usar dois mecanismos para obter referencias de objetos remotos. Ela pode registrar o objeto remoto com a ferramenta de nomes do RMI, que se chama “rmiregistry”, ou ela pode passar e retornar referencias aos objetos como parte de sua operação normal
* Se comunicar com objetos remotos – os detalhes de comunicação entre objetos remotos são tratados pelo RMI, ou seja, para o programador, a comunicação remota é semelhante a uma chamada ao método localmente.
* **Carregar “bytecodes” de objetos móveis – como o RMI permite que objetos remotos sejam passados com o parâmetro numa função, ele fornece os mecanismos necessários para carregar o código dos objetos remotos.

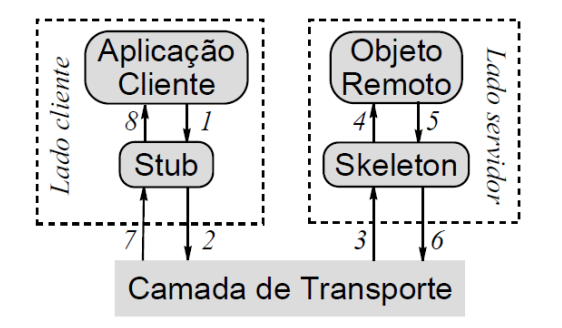
*A Figura ilustra como uma aplicação RMI distribuída usa o “rmiregistry” para obter uma referencia ao objeto remoto. O servidor chama o registro para associar um nome ao objeto remoto. O cliente procura o objeto remoto pelo seu nome no registro do servidor e então invoca um método nele. A ilustração também mostra o RMI utilizando um servidor “web” para carregar os “bytecodes”, de servidor para cliente e vice-versa, de acordo com as necessidades da aplicação.*

Para um melhor entendimento, estão descritos em ordem os passos que acontecem na comunicação, do inicio da invocação de um método por um cliente, até o momento em que a aplicação recebe os valores de retorno.

São estes os passos:

• Surge a necessidade da aplicação cliente utilizar um serviço disponibilizado pelo servidor. Esta necessidade é passada ao stub que ficará responsável por toda a comunicação e decodificação dos parâmetros envolvidos.

• O stub inicia uma conexão com o servidor em que o objeto remoto está localizado. As informações que precisam ser enviadas ao servidor são colocadas no parâmetro da invocação e serializadas pelo stub para posterior envio. A invocação está pronta e então é transmitida ao servidor pela rede.

 • O servidor cria um elemento denominado skeleton que ficará responsável por receber as invocações que chegam do stub.

• Logo após receber a invocação, o skeleton decodifica os parâmetros e repassa ao servidor para que este execute o método que irá atender a necessidade da aplicação cliente.

• O método é executado e o resultado é passado ao skeleton para que este faça a comunicação corresponde à mensagem de retorno.

• Antes de enviar a resposta ao cliente, o skeleton serializa os parâmetros da resposta e então a transmite pela rede.

• O stub recebe a resposta enviada pelo skeleton.

• Após receber, o stub decodifica os parâmetros da mensagem de reposta e repassa a aplicação cliente para que esta utilize o resultado do serviço utilizado.

## Serialização

A capacidade de armazenar e recuperar objetos em Java é essencial quando se fala de RMI. A maneira encontrada para armazenar esses objetos em uma forma serializada sem ocupar muita memória foi representar os estados do objeto suficientes para reconstruí-lo.

Objetos serializados em Java devem ser capazes de identificar e verificar a classe Java da qual eles derivam e recuperar seu conteúdo para criar uma nova instância.

Freqüentemente, objetos armazenados fazem referência a outros objetos. Esses outros objetos também devem ser armazenados e na recuperação, todos devem ser dezarmazenados ao mesmo tempo, para que as relações entre eles não se percam

## Stubs e Skeletons

RMI usa um mecanismo padrão empregado em sistemas RPC, para se comunicar com objetos remotos, que são “stubs” e “skeletons”. O “stub” funciona semelhante a um proxy para o objeto remoto. Quando um objeto local invoca um método num objeto remoto, o “stub”fica responsável por enviar a chamada ao método para o objeto remoto[DEITEL e DEITEL,2001].

Passos do “stub” quando é invocado:

* Iniciar conexão com a “Virtual Machine” que contém o objeto remoto.
* Escrever e transmitir os parâmetros para a “Virtual Machine” remoto.
* Esperar pelos resultados da invocação do método.
* Ler os resultados retornados.
* Retornar os valores ao objeto que executou a chamada.

O “stub” esconde a serializacao dos parâmetros e toda a comunicação a nível de rede, com o objetivo de simplificar o mecanismo de realização da chamada.

Na “Virtual Machine” remota, cada objeto deve ter um “skeleton” correspondente ao “stub”. O “skeleton” é responsável por enviar a chamada ao objeto remoto.

Passos do “skeleton” quando recebe uma chamada:

* Ler os parâmetros enviados pelo “stub”.
* Envocar o método no objeto remoto.
* Escrever e transmitir o resultado ao objeto que executou a chamada.

## Ativação

Sistemas de objetos distribuídos são projetados para suportar objetos persistentes de vida longa. Tendo em vista que tais sistemas podem conter milhares de objetos, não se torna razoável que esses objetos se tornem ativos e continuem sempre ativos, consumindo assim recursos preciosos do sistema. Além disso, clientes devem armazenar referências persistentes aos objetos, para que a comunicação possa ser restabelecida depois de uma queda no sistema, considerando que a referência a um objeto só é válida enquanto este estiver ativo.

Ativação em Java é um mecanismo que provê referências persistentes aos objetos, ao mesmo tempo em que gerencia a execução do mesmo. Em RMI, a ativação permite que objetos iniciem sua execução a medida que esta ação é necessária.

Quando ativado, o objeto remoto é acessado, caso ele não esteja sendo executado, e o sistema inicia a execução do objeto dentro da “Java Virtual Machine” apropriada.

## Interface

Primeiro deve se criada a interface do servidor, nesta interface estarão os métodos remotos que o cliente utilizara para interagir com o servidor.

Exemplo:

*public interface Teste extends Remote {*

*public void get( ) throws RemoteExeception;*

*}*

Segundo deve implementar um servidor remoto RMI. Normalmente a declaração de um classe que implementa um servidor e da seguinte forma:

Como foi declarado o método *get()* na interface, deve-se implementar esse método na classe TesteImp, pelo motivo que esta classe é a implementação da classe Teste e seus métodos.

*public class TesteImp extends UnicastRemoteObject implements*

*Teste*

*{*  *public TesteImp( )*

*{*  *super( );*

*}*

*public void get( ) throws RemoteException*

*{*

*System.out.println(“O flamengo é bom!!!”);*

*}*

*public static void main( String args[] ) thorows Exception*

*{*

*TesteImp tmp = new TesteImp( );*

*String serveObjectName = “/127.0.0.1/Serv”;*

*// atribuir o nome do servidor ao rmiregistry*

*Naming.rebind( serveObjectName, tmp);*

*System.err.println(“Servidor não esta funcionando!!!”);*

*}*

*}*

Esta classe deve implementar todos os métodos declarados na interface do servidor.

Uma vez que a interface está definida e a classe que implementa o serviço remoto tenha sido criada, o próximo passo no desenvolvimento da aplicação distribuída é desenvolver o cliente.

*public class cliente {*

*public static void main ( String args[] 0 {*

*try {*

*TesteIpm objre = ( testeimp ) Naming.lookup(“//127.0.0.1/Serv”);*

*objre.get( );*

*}*

*catch ( java.rmi.ConnectException ec )*

*{*

*System.out.println(“Conexão com servidor falhou”);*

*}*

*catch ( Exception e)*

*{*  *e.printStackTrace( );*  *System.exit( 1 );*

*}*

*}*

*}*

Na implementação da classe cliente, na linha 4, existirá uma declaração de um objeto remoto do servidor que estará localizado na url “*//127.0.0.1/Serv*”.