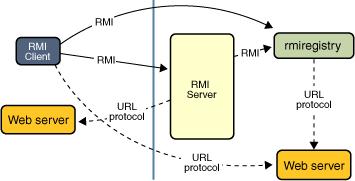
**Aula Prática de Sistemas Distribuídos – RMI**

A figura 1 abaixo apresenta uma aplicação RMI sedo originada através da publicação (registro) do objeto no rmiregistry. Através do Web Server o RMICliente faz requisições aos objetos remotos, que consequentemente respondem a solicitação

****

**Figura 1 – RMI –(http://docs.oracle.com/javase/tutorial/rmi/overview.html)**

**RMI Registry**

Em uma máquina host, um programa servidor cria um serviço remoto, primeiramente criando o objeto que implemente aquele serviço. No nosso caso o testeola.java

Em seguida ele exporta aquele objeto para o RMI. Quando o objeto é exportado, o RMI cria um serviço que aguarda as conexões do cliente. O servidor registra o objeto no RMI Registry, com um nome público. No nosso caso “OlaServidor”

No lado do cliente o RMI Registry é acessado através da classe estática Naming. Ela provém o método lookup( ), que o cliente usa para requisitar o registro. Esse método aceita a URL que especifica o nome do servidor e o nome do serviço desejado.

O método retorna uma referência remota para o objeto do serviço. A URL é formada como a seguir:

rmi://[host\_name]:[port\_number]/[service\_name]

**WEB Server**

**"Stubs"**

O "stub" funciona semelhante a um proxy para o objeto remoto. Quando um objeto local invoca um método em um objeto remoto, o "stub" fica responsável por enviar a chamada ao método para o objeto remoto.

Passos do "stub" quando é invocado:

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

**"Skeletons"**

Na Máquina Virtual 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".
* Invocar o método no objeto remoto.
* Escrever e transmitir o resultado ao objeto que executou a chamada.

Programas a serem implementados. (Caso não esteja configurado)

Setar o classpath do windows ( Sistema, Configurações Avançadas, Variáveis de ambiente)

Nome Variável: JAVA\_HOME

Valor da Variável: C:\Arquivos de programas\Java\jdk1.6.0\_25

Nome Variável: CLASSPATH

Valor da Variável:

.;%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\htmlconverter.jar;%JAVA\_HOME%\jre6\lib;%JAVA\_HOME%\jre6\lib\rt.jar

Crie cada um destes programas em bloco de notas com seus respectivos nomes.

testeola.java

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface testeola extends Remote {

String showMsg(String msg) throws RemoteException;

}

OlaImpl.java

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

public class OlaImpl extends UnicastRemoteObject implements testeola {

// precisamos definir um construtor pois o default não gera RemoteException

public OlaImpl() throws RemoteException {

super();

}

public String showMsg(String msg) {

// note que não tem o throws aqui. A exceção é gerada pela super-classe se for o caso

System.out.println("msg: " + msg);

return("msg enviada- o cliente tem notificação de recebimento");

// caso queira passar algum parâmetro para o cliente.

}

public static void main(String args[]) {

try {

// aqui criamos o objeto que sera acessado remotamente

OlaImpl obj = new OlaImpl();

// Aqui registramos o objeto obj como "OlaServidor" no servidor, da-se um nome a interface

Naming.rebind("OlaServidor", obj);

System.out.println("Servidor carregado no registry");

} catch (Exception e) {

System.out.println("OlaImpl erro: " + e.getMessage());

e.printStackTrace();

}

}

}

Cliente.java

import java.rmi.Naming;

public class Cliente {

public static void main(String[] args) {

/\*

\* obj é o identificador utilizado para fazer referência ao objeto remoto que será implementado

\*/

testeola obj = null;

String msg = "aqui esta a mensagem que será passada para o servidor";

String retorno = null;

try {

/\*

\* - o lookup carrega o OlaImpl\_Stub do CLASSPATH

\* - 127.0.0.1 é o IP da máquina onde está o servidor, no nosso caso é a mesma máquina, mas pode ser a máquina do colega

\* - OlaServidor é o nome que utilizamos para fazer referência ao objeto no servidor

\*/

// aqui instanciamos o objeto remoto

obj = (testeola)Naming.lookup("//127.0.0.1/OlaServidor");

// agora executamos o método showMsg no objeto remoto

retorno = obj.showMsg(msg);

System.out.println(retorno);

}

// Aqui trata-se as exceções presentes

catch (Exception e) {

System.out.println("Client exception: " + e.getMessage());

e.printStackTrace();

}

}

}

Agora vamos compilar os arquivos. No diretório. - javac \*.java

Agora vamos criar uma classe do tipo Stub para a classe que será acessada remotamente. - rmic OlaImpl

Iniciar o servidor RMI. - start rmiregistry

Abrir prompt’s para o servidor e para o cliente

Trabalho prático em grupo. – Valor 7,5 pts.

Desenvolver uma aplicação distribuída que faça acesso remoto a objetos de um banco de dados permitindo a consulta e a manipulação destas variáveis.

A aplicação deve prever acessos simultâneos ao servidor.

Cada grupo deve pensar em uma aplicação propondo as características e desenvolvendo a aplicação e o teste dos resultados.

Dê preferência pelo desenvolvimento em Java.

O Trabalho deve ser entregue em um arquivo zipado contendo:

- uma capa com o nome dos Integrantes, além é claro da discussão das características apresentadas pela aplicação desenvolvida e seu foco de aplicação. (1pts)

- Arquivos compilados e em prefeito funcionamento, bem como os arquivos originais de desenvolvimento. (5,5pts)

- Uma explicação sobre o funcionamento da aplicação e suas funcionalidades empregadas para alcançar o objetivo. (1pts)