TRABALHO JAVA RMI

Tutorial – Desenvolvendo um Servidor RMI

Para aceitar chamadas remotas de métodos via RMI, um servidor deve estender a interface java.rmi.Remote e declarar os métodos que serão acessados remotamente.

Na interface RMI abaixo é definido um método sayHello(), que retorna uma saudação quando é chamado.

```
/** HelloInterface.java **/
import java.rmi.*;

public interface HelloInterface extends Remote {
   public String sayHello() throws RemoteException;
}
```

A exceção java.rmi.RemoteException indica erros na chamada remota, e deve ser prevista pelos métodos de interfaces RMI.

Agora vamos implementar a interface. Para facilitar o nosso trabalho, vamos usar como base a classe UnicastRemoteObject, que já implementa alguns métodos necessários para o servidor. Temos que criar também um construtor para o nosso servidor (neste caso, ele apenas chama o construtor da classe base).

```
/** HelloServer.java **/

import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class HelloServer extends UnicastRemoteObject implements HelloInterface {
   public HelloServer() throws RemoteException{ super() }
   // main()
   // sayHello()
}
```

No método main() do servidor, iremos criar um objeto que implementa a interface HelloInterface e registrá-lo como um servidor no registro do RMI, com o nome "Hello", para que ele possa ser localizado pelos clientes.

```
   public static void main(String[] args) {
      try {

         // Instancia o objeto servidor e o seu stub
         HelloServer server = new HelloServer();


         // Registra o stub para que possa ser obtido pelos clientes
         Registry registry = LocateRegistry.createRegistry(5099);
         registry.bind("Hello", server);
         System.out.println("Servidor pronto");

      } catch (RemoteException | AlreadyBoundException ex) {
         System.err.println(ex);
      }
   }
```

Temos que implementar agora os métodos definidos na interface do servidor. Nesta aplicação, há apenas o método sayHello() para ser implementado.

```
public String sayHello() throws RemoteException {
   System.out.println("Executando sayHello()…");
   return "Hello!";
}
```

Criando um Cliente RMI

Crie um cliente que obtenha uma referência para o servidor no registro RMI e chame o método sayHello().

```
/** HelloClient.java **/

import java.rmi.registry.*;

public class HelloClient {

   public static void main(String[] args) {

      try {
         // Obtém o stub do servidor
         HelloInterface stub = (HelloInterface)
                Naming.lookup("rmi://localhost:5099/Hello");

         // Chama o método do servidor e imprime a mensagem
         String msg = stub.sayHello();
         System.out.println("Mensagem do Servidor: " + msg);

      } catch (RemoteException | NotBoundException ex) {
         System.err.println(ex);
      }
   }

}
```

Trabalho

1. Crie uma aplicação RMI de agenda de telefones que mantém um arquivo de nomes e números de telefone. Defina a interface *PhoneBookServer* com os seguintes métodos:

```
public ArrayList<PhoneBookEntry> getPhoneBook()
public void addEntry(PhoneBookEntry entry)
```

   - A classe *PhoneBookServerImpl* deve implementar a interface *PhoneBookServer*.
   - A classe *PhoneBookEntry* deve conter atributos que representam o primeiro nome, o sobrenome e o número de telefone de uma pessoa. A classe também deve fornecer métodos get/set adequados. A classe deve implementar *Serializable* para que os objetos possam ser serializados.
   - A classe *PhoneBookClient* deve permitir adicionar um registro e listar os registros existentes.

Escrever um **relatório** (modelo artigo 4C) descrevendo o que foi desenvolvido e explicando o código criado.

Entrega: dia 14/04 pelo Canvas

Nos dias 31/03 e 07/04 nos horários em que teríamos aula estarei disponível para **tirar dúvidas** via email (maria.villarreal@ifc.edu.br). Dúvidas podem ser enviadas para o mesmo email em outros horários, porém sem garantia de que serão respondidos imediatamente.