

Trabalho 2 - Sistemas Distribuídos - Remote Method Invocation (RMI)

Francisco Romes da Silva Filho, 409976

Repositório com solução, acesse a pasta que se refere ao número do trabalho:

<<https://github.com/romesdev/sd-2020-2>>

PARTE 1) Crie uma aplicação, cliente/servidor usando o serviço RMI (Invocação Remota de Método).

O pacote que se refere a essa solução é: caseRemoto.

Nessa implementação temos 4 classes do tipo POJO:

- Pessoa que tem um carro;
- Motorista, uma herança de Pessoa;
- Carro que pode ou não ter um dono (Motorista);
- Caneta;

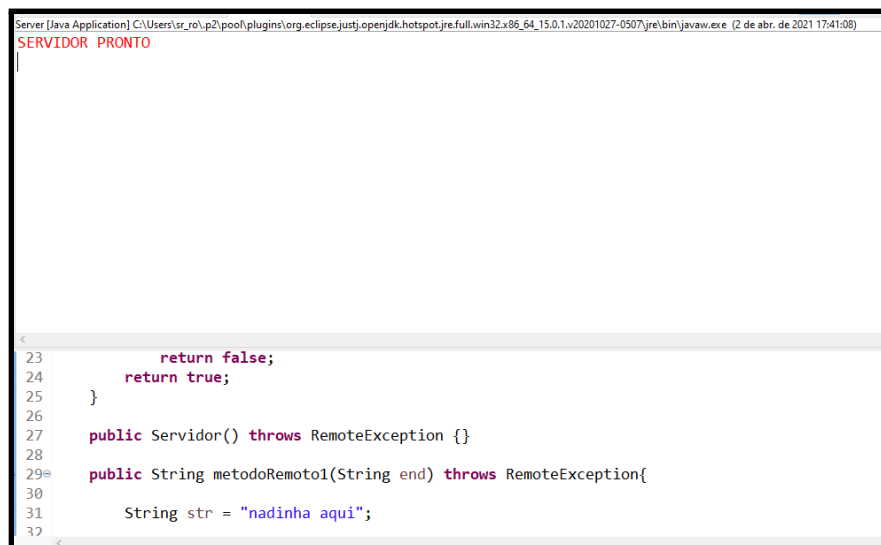
A implementação usa um cliente simples para fazer uma invocação de até 4 métodos remotos para um servidor que pode estar sendo executado em um host remoto. O cliente recebe o retorno de um dos 4 métodos implementados do servidor. No exemplo de execução, os 4 métodos são executados para efeitos de demonstração.

A implementação tem uma Interface implementada pelo Servidor (ele implementa os 4 métodos). O cliente realiza a chamada dos métodos.

No Servidor, é criado um registro em uma porta definida, com *“LocateRegistry.createRegistry(port)”* ou, então, pegar o atual. E usando *“bind”* ou *“rebind”* para alterar a ligação do nome de registro.

No Cliente, com o registro resgatado do Servidor é usado o lookup, que retorna uma referência, um stub, para o objeto remoto associado ao nome (endereço passado) especificado, neste registro. Usando: *Naming.lookup("//localhost:"+ porta+"/InterfaceRemota")*. Portanto, finalmente, o Cliente tem acesso a InterfaceRemota e pode invocar os métodos.

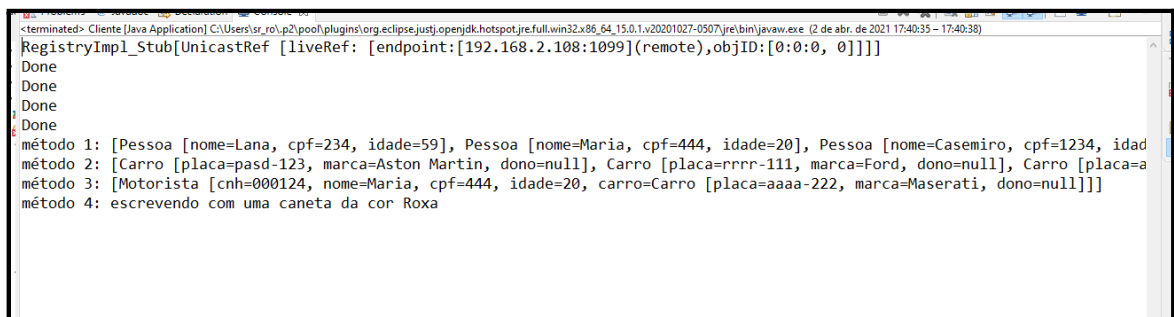
Exemplo de Servidor Pronto (rodando)



```
Server [Java Application] C:\Users\sr_ro\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.15.0.1.v20201027-0507\jre\bin\javaw.exe (2 de abr. de 2021 17:41:08)
SERVIDOR PRONTO

23         return false;
24         return true;
25     }
26
27     public Servidor() throws RemoteException {}
28
29     public String metodoRemoto1(String end) throws RemoteException{
30
31         String str = "nadinha aqui";
32     }
```

Execução no lado do Cliente (resposta dos 4 métodos)



```
<terminated> Cliente [Java Application] C:\Users\sr_ro\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.15.0.1.v20201027-0507\jre\bin\javaw.exe (2 de abr. de 2021 17:40:35 - 17:40:38)
RegistryImpl_Stub[UnicastRef [liveRef: [endpoint:[192.168.2.108:1099](remote),objID:[0:0:0, 0]]]]
Done
Done
Done
Done
método 1: [Pessoa [nome=Lana, cpf=234, idade=59], Pessoa [nome=Maria, cpf=444, idade=20], Pessoa [nome=Casemiro, cpf=1234, idade=20]]
método 2: [Carro [placa=pasd-123, marca=Aston Martin, dono=null], Carro [placa=rrrr-111, marca=Ford, dono=null], Carro [placa=aaaa-222, marca=Maserati, dono=null]]
método 3: [Motorista [cnh=000124, nome=Maria, cpf=444, idade=20, carro=Carro [placa=aaaa-222, marca=Maserati, dono=null]]]
método 4: escrevendo com uma caneta da cor Roxa
```

PARTE 2) Sobre a representação externa de dados e serialização crie uma aplicação cliente/servidor com Socket com no mínimo dois métodos.

O pacote que se refere a essa solução é: sockets.

Seguindo a orientação de protocolo requisição-resposta do livro texto. Protocolo esse que é baseado nas operações *doOperation*, *getRequest* e *sendReply*. O Cliente implementa a

operação *doOperation* que realiza uma requisição e fica esperando por uma resposta do Servidor que implementa as operações *getRequest*, que fica atento a qualquer requisição feita ao Servidor, e *sendReply*, que realiza o envio da resposta (reply).

Para execução deste protocolo é preciso ainda a implementação de mais duas classes sugeridas pelo autor do livro texto: *MensagemRequisição* e *ObjectRemoteReference*.

MensagemRequisição: Guarda as informações essenciais para a realização das transações, como tipo de operação, método requisitado, referência do objeto remoto, argumentos (em bytes), id de requisição. Informações serializadas e convertidas para bytes.

ObjectRemoteReference: simulando um objeto remoto que para ser invocado em outro ambiente. Quando tal objeto é invocado, os seus argumentos são empacotados e enviados a partir da máquina virtual local para o remoto, onde os argumentos são desempacotados e usados.

O Servidor, ativo, pega as requisições (*getRequest*), desempacota, trata o método a ser invocado, realiza a chamada e, finalmente, faz o envio (*sendReply*). Os métodos chamados podem ser *metodo1* - que realiza uma concatenação de exemplo com uma mensagem padrão - ou *metodo2* - que realiza a inversão da String passada. É nessa parte que a reflexão, age delimitando os métodos e argumentos que serão executados de uma classe que não é conhecida em tempo de execução.

O Cliente, esperando, recebe o retorno do Servidor (do *sendReply*) e mostra o resultado. Portanto, mostra o objeto retornado na transmissão.

Exemplificando a execução da aplicação

Considerando que o arquivo de execução (arquivo *Run.java*) do Servidor foi executado primeiro, manteve-se rodando (e não teve retentativas, para não ocasionar procura pela mesma porta). O funcionamento da aplicação se resume a sequências de fotos, com a execução do Cliente (arquivo *Operação.java*), a seguir.

1) Servidor Pronto, rodando na porta 1928.



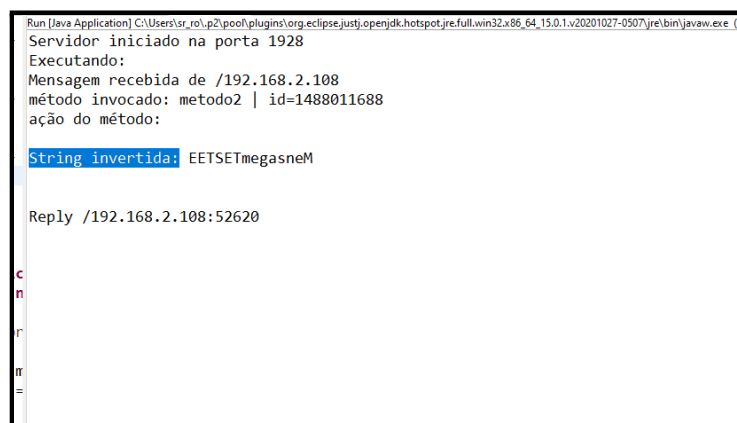
- 2) **Cliente fazendo uma requisição (doOperation) e rapidamente sendo respondido. Perceba que o Servidor enviou uma mensagem em forma de objeto como retorno.**



```
<terminated> Client [Java Application] C:\Users\sr_ro\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v20201027-0507\jre\bin\javaw.exe (7 de abr. de 2021 14:17:23 - 14:17:23)
Requisição
Começo da operação...

O que foi recebido:
MensagemRequisicao [type=1, remoteObjectRef=null, requestId=0, args=[-84, -19, 0, 5, 117, 114, 0, 19, 91,
```

- 3) **Observando, novamente, o Servidor após a requisição do Cliente, percebemos que ele pegou a requisição (getResquest), tratou, executou o método e, então, retornou a resposta (sendReply). Observe que o método 2 foi executado, ou seja, a mensagem passada como parâmetro “MensagemTESTEE” foi invertida na operação com o Servidor.**



```
Run [Java Application] C:\Users\sr_ro\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v20201027-0507\jre\bin\javaw.exe
Servidor iniciado na porta 1928
Executando:
Mensagem recebida de /192.168.2.108
método invocado: metodo2 | id=1488011688
ação do método:
String invertida: EETSETmegasneM

Reply /192.168.2.108:52620
```

Referências

Sistemas distribuídos [recurso eletrônico] : conceitos e projeto / George Coulouris ... [et al.] ; tradução: João Eduardo Nóbrega Tortello ; revisão técnica: Alexandre Carissimi. – 5. ed. – Dados eletrônicos. – Porto Alegre : Bookman, 2013.

Getting Started Using Java RMI.

Link:<<https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html>>

Java Reflection: um exemplo prático.

Link:<<http://blog.gabrielamorim.com/java-reflection-um-exemplo-pratico/>>