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;

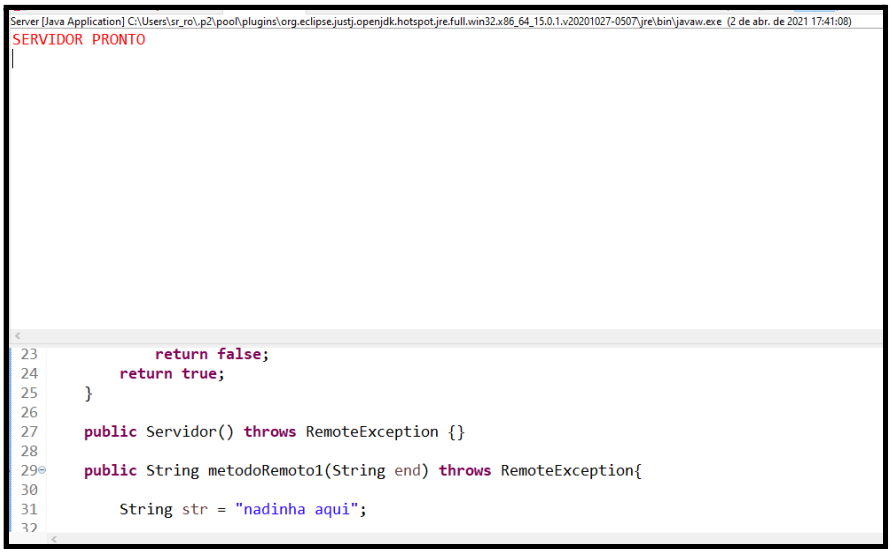
A implementação usa um cliente simples para fazer uma invocação de até 4 métodos remotos para um servidor que pode estar sendo executado em um host remoto. O cliente recebe o retorno de um dos 4 métodos implementados do servidor. No exemplo de execução, os 4 métodos são executados para efeitos de demonstração.

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

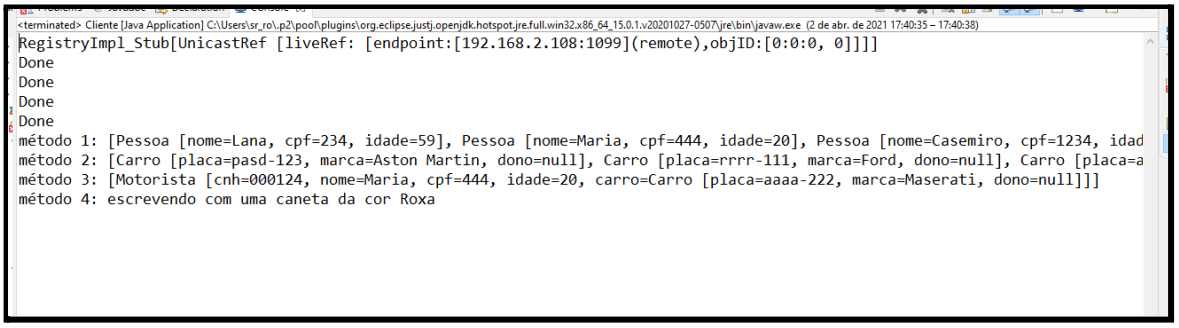
No Servidor, é criado um registro em uma porta definida, com *"LocateRegistry.createRegistry(port)*" ou, então, pegar o atual. E usando *"bind"* ou *"rebind"* para alterar a ligação do nome de registro.

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)



Execução no lado do Cliente (resposta dos 4 métodos)



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.

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/>