UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO E CIÊNCIAS DA COMPUTAÇÃO

ReChat

João Pedro de A. Paula

Paulo Lima

Natal - RN

Julho, 2021

1 Introdução

Este projeto consiste na implementação de um sistema de chat distribuído utilizando as bibliotecas de Remote Method Invocation (RMI) da linguagem Java.

O mesmo foi solicitado como projeto da disciplina de Programação Distribuída.

2 Arquitetura, Implementação e Execução

2.1 Arquitetura

O projeto foi arquiteturado de modo que a interface de métodos e comunicação de ambos os lados, cliente e servidor, fosse a mesma, permitindo que o servidor também possa agir como um cliente, e que os clientes tenham acesso a lista de usuários tal qual o servidor.

2.2 Implementação

O projeto se dividiu em 4 classes. A classe `Protocol` é uma interface que define os métodos que tanto o cliente como o servidor precisam implementar. Esta classe extende a classe `Remote`, da biblioteca do Java RMI, de modo a permitir e facilitar a utilização da mesma como parte de um servidor RMI. A classe `Chat` é a classe responsável pela implementação dos métodos da classe `Protocol`. E, por fim, temos as duas classes que implementam o servidor e o cliente, ambas com um método main, chamadas, respectivamente, `Server` e `Client`. A seguir, uma imagem da interface `Protocol`.

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface Protocol extends Remote {
    public void send(String msg) throws RemoteException;

    public String getName() throws RemoteException;

    public List<Protocol> getClients() throws RemoteException;

    public void newClient(Protocol client) throws RemoteException;

    public void removeClient(Protocol client) throws RemoteException;

    public int getMessageNumber() throws RemoteException;

    public void updateClients() throws RemoteException;
}
```

A seguir se encontra a implementação dos métodos que correspondem a lógica do chat, presentes na class `Chat`.

```java
public void send(String msg) throws RemoteException {
    System.out.println(msg);
    messageNumber++;
}

public void newClient(Protocol client) throws RemoteException {
    send("<" + client.getName() + "> has entered the chat!");
    for (Protocol c : clients) {
        c.send("<" + client.getName() + "> has entered the chat!");
    }
    clients.add(client);
}

public void removeClient(Protocol client) throws RemoteException {
    send("<" + client.getName() + "> left the chat!");
    for (Protocol c : clients) {
        c.send("<" + client.getName() + "> left the chat!");
    }
    clients.remove(client);
}

public void updateClients() throws RemoteException {
    List<Protocol> aux = clients;
    for (Protocol client : aux) {
        if (client.equals(null)) {
            removeClient(client);
        }
    }
}
```

2.3 Execução

Para executar o servidor, basta abrir um terminal e se dirigir a pasta onde os arquivos do projeto se encontram e digitar `make run-server`, e o Makefile se encarregará de compilar as classes do servidor e executar a main do mesmo. A seguir, o usuário é solicitado para inserir um nome para o servidor e logo após este já está funcional. É essencial que o servidor seja executado antes que qualquer cliente.

Para a execução do código do cliente, é necessário, em um outro terminal, se dirigir a pasta do projeto e executar `make run-client`, onde o

usuário será solicitado para inserir um nome de usuário e, em seguida, caso o nome de usuário esteja disponível, poderá começar a digitar mensagens, sempre apertando Enter para enviá-las ao servidor e demais clientes. Caso o nome de usuário já esteja sendo utilizado, o usuário será solicitado para inserir um novo nome.

3 Conclusão

Pôde-se observar no decorrer do trabalho que as aplicações de comunicação via chat se prestam muito bem a programação distribuída, visto que é natural que tenhamos um protocolo que define a comunicação entre cliente e servidor, mas que estes últimos geralmente se encontram em processos, ou, mais comumente, em máquinas diferentes, de diferentes redes. Além disso, a linguagem Java, especialmente desde o Java 11, se mostra com uma API muito rica para implementação de aplicações com RMI, abstraindo muito do código boilerplate associado a esses tipos de programas. A experiência foi definitivamente enriquecedora, nos fazendo compreender de forma mais prática a utilização de aplicação de sistemas distribuídos com base na execução remota de procedimentos.

Referências

DAVI. Uma introdução ao RMI em Java. 2013. Disponível em: https://www.devmedia.com.br/uma-introducao-ao-rmi-em-java/28681. Acesso em 02 de Julho de 2021.

FIORESI, Cristiano. Tutorial RMI - Remote Method Invocation. 2007. Disponível em: https://www.devmedia.com.br/tutorial-rmi-remote-method-invocation/6442. Acesso em 02 de Julho de 2021.