Engenharia de Telecomunicações
STD29006 – Sistemas Distribuídos

Prof. Emerson Ribeiro de Mello
mello@ifsc.edu.br

Laboratório 3.1: Java RMI

24/11/2021

1 Organização dos arquivos do projeto

1.1 Projeto no lado do servidor

```
.
`-- std29006
    |-- ContadorDistribuido.java
    `-- server
        |-- Contador.java
        `-- Servidor.java
```

1.2 Projeto no lado do cliente

```
.
`-- std29006
    |-- client
    |   `-- Cliente.java
    `-- ContadorDistribuido.java
```

2 Criando a interface do objeto distribuído

Na interface são descritos dois métodos: `void incrementar()` e `int obterValorAtual()`. Esse arquivo Java deve ser compartilhado entre o servidor e as aplicações clientes. Note que o nome do pacote Java deve ser respeitado por servidor e clientes. Tem que ter o mesmo nome.

```java
package std29006;
import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 * Interface que deve ser compartilhada por servidor e clientes
 */
public interface ContadorDistribuido extends Remote{

    public void incrementa() throws RemoteException;
    public int obtemValorAtual()  throws RemoteException;
}
```

3 Desenvolvendo a aplicação servidora

Na listagem abaixo é apresentada a estrutura de diretórios e arquivos do projeto Java do servidor.

A classe `Contador.java` implementará a interface `ContadorDistribuido.java`.

```java
package std29006.server;
import std29006.ContadorDistribuido;
/**
 * Classe que implementa a interface do objeto distribuído
 */
public class Contador implements ContadorDistribuido{
    private int valor = 0;
    @Override
    public void incrementa() throws RemoteException {
        this.valor++;
    }
    @Override
    public int obtemValorAtual() throws RemoteException {
        return this.valor;
    }
}
```

A classe `Servidor.java` é responsável por instanciar um objeto da classe `Contador`, subir um serviço de nomes (*rmiregistry*) e registrar a instância da classe Contador no serviço de nomes.

```java
package std29006.server;
import std29006.ContadorDistribuido;
import java.rmi.AlreadyBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 * Classe responsável por criar uma instância do objeto Contador e registrá-la
 * em um serviço de registro de objetos distribuídos
 */
public class Servidor {

    // Constantes que indicam onde está sendo executado o serviço de registro,
    // qual porta e qual o nome do objeto distribuído
    private static String nomeServidor = "127.0.0.1";
    private static int porta = 12345;
    private static final String NOMEOBJDIST = "MeuContador";

    public static void main(String args[]){
        try {
            // recebendo nome do servidor por argumento de linha de comando
            if (args[0] != null){
                nomeServidor = args[0]
            }

            // recebendo porta do rmiregistry por argumento de linha de comando
            if (args[1] != null){
                porta = Integer.parseInt(args[1]);
            }

            // Criando
            Contador c = new Contador();

            // Definindo o hostname do servidor
            System.setProperty("java.rmi.server.hostname", nomeServidor);

            ContadorDistribuido stub = (ContadorDistribuido)
                                       UnicastRemoteObject.exportObject(c, 0);
```

```
            // Criando serviço de registro
            Registry registro = LocateRegistry.createRegistry(porta);

            // Registrando objeto distribuído
            registro.bind(NOMEOBJDIST, stub);

            System.out.println("Servidor pronto!\n");
            System.out.println("Pressione CTRL + C para encerrar...");


        } catch (RemoteException | AlreadyBoundException ex) {
            Logger.getLogger(Servidor.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

4 Desenvolvendo aplicativo Cliente

A classe `Cliente.java` é responsável por invocar os métodos remotos do Servidor. Ao executar o cliente é necessário informar o nome (ou IP) e porta da máquina onde o *rmiregistry* está sendo executado.

```java
package std29006.client;
import std29006.ContadorDistribuido;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Cliente de uma aplicação Java RMI
 */
public class Cliente {
    private static String nomeServidor = "127.0.0.1";
    private static int porta = 12345;
    private static final String NOMEOBJDIST = "MeuContador";

    public static void main(String args[]) {
        try {
            if (args[0] != null){
                nomeServidor = args[0];
            }
            if (args[1] != null){
                porta = Integer.parseInt(args[1]);
            }
            System.out.println("Conectando no servidor "+ nomeServidor);

            // Obtendo referência do serviço de registro
            Registry registro = LocateRegistry.getRegistry(nomeServidor, porta);

            // Procurando pelo objeto distribuído registrado previamente com o NOMEOBJDIST
            ContadorDistribuido stub = (ContadorDistribuido) registro.lookup(NOMEOBJDIST);

            // Invocando métodos do objeto distribuído
            System.out.println("Valor atual: " + stub.obtemValorAtual());
            System.out.println("Solicitando ao servidor para incrementar o contador");
            stub.incrementa();
            System.out.println("Valor atual: " + stub.obtemValorAtual());

            System.out.println("Fim da execução do cliente!");

        } catch (RemoteException | NotBoundException ex) {
            Logger.getLogger(Cliente.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

5 Compilando e executando o projeto

5.1 Servidor

No servidor é necessário criar o arquivo (java.policy) com a seguinte política de segurança:

```
grant{
    permission java.security.AllPermission;
}
```

```
javac std29006/server/Servidor.java
java -Djava.security.policy=java.policy std29006.server.Servidor 127.0.0.1 12345
```

5.2 Cliente

```
javac std29006/client/Cliente.java
java std29006.client.Cliente 127.0.0.1 12345
```

6 Exercício: Sistema de inventário de ativos de WiFi

Desenvolva uma aplicativo cliente / servidor baseado em RMI para fazer o inventário de pontos de acesso (*Access Point* – AP) WiFi. Ao cadastrar um AP é necessário informar um nome, *MAC address*, frequências que opera (2.4Ghz, 5Ghz ou ambas) e em qual sala do prédio esse AP está afixado (por exemplo: lab. redes II). O aplicativo servidor é o responsável por manter esse inventário em memória e o aplicativo cliente é responsável por fazer requisições ao servidor para cadastrar, remover ou listar pontos de acesso. Ao listar os pontos de acesso o cliente poderá indicar critérios de seleção. Exemplos:

- Cadastrar um novo AP na sala lab SiDi e que opere em ambas as frequências

  ```
  java ClienteInventario <endereço servidor> add "AP1" "mac:abcdef" "freq:2.4:5" "Lab SiDi"
  ```

- Listar todos os APs

  ```
  java ClienteInventario <endereço servidor> list
  ```

- Listar todos os APs que operam com a frequência de 5Ghz.

  ```
  java ClienteInventario <endereço servidor> list freq:5
  ```

- Remover AP com o identificador AP2

  ```
  java ClienteInventario <endereço servidor> del AP2
  ```