Universidade da Beira Interior

Sistemas Distribuídos
Licenciatura em Engenharia Informática

Frequência
2013/06/04

Duração: 1 hora e 50 minutos

(10 valores)

I

Nota: Nos exercícios 1 e 2, pode omitir o tratamento de exceções.

1 – A classe Cliente, apresentada na página seguinte, permite comunicar por Sockets com um processo servidor. Esse servidor contém uma lista de nomes, a implementar através de um objeto do tipo java.util.Vector. O processo cliente permite escolher as opções seguintes: 1 – Inserir um nome na lista de nomes, obtendo como resultado se a inserção teve ou não sucesso (uma inserção ser efetuada com sucesso significa que o nome ainda não existia na lista); 2 – Nº de Cliente, que devolve o número de quantos processos clientes já se ligaram ao servidor até ao momento em que esse processo se ligou; 3 – Sair, que termina a execução do processo cliente.

- Construa o Servidor ao qual o processo cliente apresentado (ver listagem 1) possa aceder. O processo servidor deverá poder atender vários clientes em simultâneo, isto é, ser um servidor "multi-threaded", em que as várias Threads podem aceder simultaneamente ao mesmo Vector de nomes. Deverá garantir a correção desse acesso simultâneo.

2 – Suponha agora que quer construir uma aplicação do mesmo tipo mas utilizando objetos distribuídos. Assim, pretende-se um processo servidor que possua um objeto remoto com os seguintes métodos: 1 – Inserir que, tal como no exercício 1, deve permitir inserir sem duplicados um nome numa lista de nomes, devolver *true* em caso de sucesso e *false* caso contrário; 2 – Nº valores, este método deverá devolver o número de valores inseridos com sucesso.
- O processo cliente deverá permitir escolher repetidamente entre as opções 1 – Inserir; 2 –Nº de valores e 3 – Sair.

- Construa a interface do objeto remoto e as classes: objeto remoto, servidor e cliente.

```
import java.io.*;
import java.net.*;
public class Cliente {
  public static void main (String [] args){
      Socket s = null;
      ObjectOutputStream os= null;
      ObjectInputStream is = null;
      int op;
      String nome=null;
      boolean sucesso = false, sair= false;
      int nCliente = 0;
      try {
            s = new Socket ("127.0.0.1", 1234);
            os = new ObjectOutputStream ( s.getOutputStream());
            is = new ObjectInputStream (s.getInputStream());
            while (!sair){
                  System.out.println
                  ("1 - Inserir nome \n2 - N° de Cliente: \n0 - Sair");
                  op = umInt();
                  os.writeInt (op);
                  os.flush();
                  switch (op){
                  case 1:
                        System.out.println("Nome:");
                        nome = umaString();
                        os.writeObject(nome);
                        os.flush();
                        sucesso= is.readBoolean();
                        System.out.println("Inserção" + sucesso);
                        break;
                  case 2:
                        nCliente = is.readInt();
                        System.out.println("Cliente n°: " + nCliente );
                        break;
                  case 0:
                        sair = true;
                        break;
                    }
          }
      }catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
      }
  }

}

  public static String umaString (){
            …  // ler uma String do teclado
  }
  public static int umInt(){
            …  // ler um int do teclado
   }
```

Listagem 1 – Código do processo Cliente

3 – Diga o que é um "callback".

4 - Descreva quais as modificações que teria de fazer para que o processo servidor do exercício 2 fizesse um "callback" ao cliente.

5 - Diga o que entende por arquitetura de um sistema distribuído.

6 – Represente esquematicamente a arquitetura das aplicações descritas nos exercícios 1 e 2.

7 – Descreva o que é o modelo de "cloud computing", e quais os diferentes tipos de serviços que fornece.

Notas auxiliares:

socket do cliente:

```
import java.net.*;
import java.io.*;

Socket meuCliente = null;
try { meuCliente = new Socket ("host", portNumber); }
catch (IOException e){ … }
```

socket do servidor:

```
 ServerSocket  meuServidor = null;
 try { meuServidor = new ServerSocket (portNumber); }
catch (IOException e){ …}

Socket sServidor = null
try { sServidor = meuServidor.accept(); }
catch (IOException e){ … }
```

RMI:

Interface remote: java.rmi.Remote

Excepção remota: java.rmi.RemoteException

Objecto remoto : java.rmi.server.UnicastRemoteObject;

Sintaxe do nome que o objecto remoto tem no RMIregistry:

[rmi:] [//] [nomeMaquina] [:port] [/nomeObjecto]

Instalar um gestor de segurança:

System.setSecurityManager (new RMISecurityManager());

Iniciar o registry: java.rmi.registry.LocateRegistry.createRegistry(1099);

Métodos da classe java.rmi.Naming:

void rebind (String nomeObjecto, Remote objecto);

Remote lookup (String nomeObjecto)

API da classe java.util.Vector:

Vector()// construtor vector vazio, dimensão inicial zero.
Vector(int capacidInicial) // construtor vector vazio, com dimensão inicial.
void addElement(Object elemento) // adiciona o elemento especificado ao final do vector.
void insertElementAt(Object obj, int indice)
// insere o elemento especificado na posição indice.

void removeElementAt(int indice) // remove o elemento na posição indice.
void setElementAt(Object obj, int indice)
// substitui o elemento da posição indice pelo objecto dado.

Object elementAt(int indice)// devolve o componente presente no indice.
void clear() // remove todos os elementos do vector.
Object clone() // devolve uma cópia do vector.
boolean contains(Object elemento)
// verifica se o objecto especificado é um componente deste vector

Object firstElement() // devolve o primeiro componente (indice 0) do vector.
Object lastElement() // devolve o último componente do vector.
int indexOf(Object elemento) // procura o índice da 1ª ocorrência de elemento
int indexOf(Object elemento, int indice) // inicia a procura anterior na posição indice.
boolean isEmpty() // verifica se o vector não tem componentes
int size() // devolve a dimensão actual.
...