Roteiro da aula

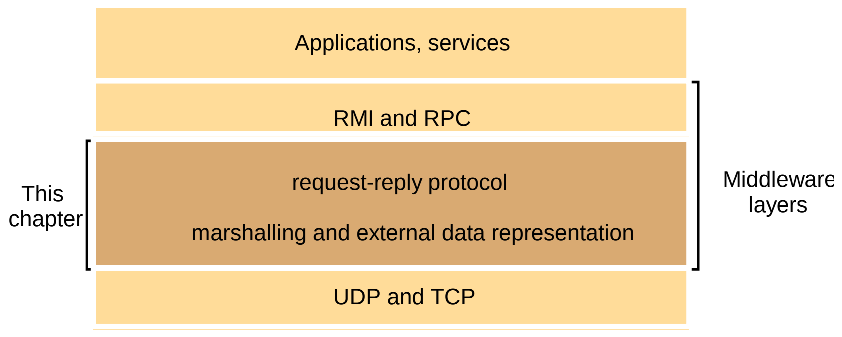
Sockets

1. UDP – (*User Datagram Protocol*)
2. TCP (*Transmission Control Protocol*)
3. IP Multicast

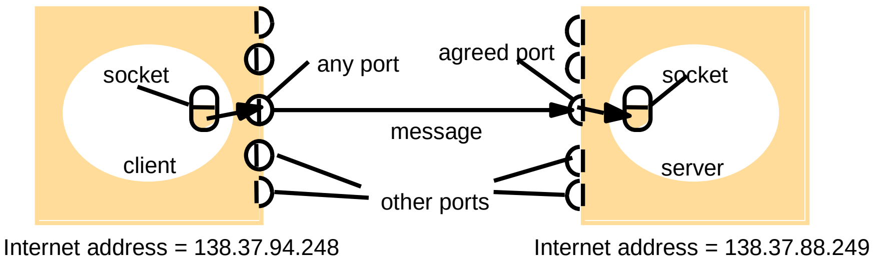
Invocação de Método Remoto usando CORBA

1. Introdução
2. Modelo de Objeto
3. Entender a Arquitetura CORBA
4. Implementar uma aplicação cliente/servidor distribuído com CORBA

Middleware layers



Sockets e Portas


socket
client
any port
agreed port
message
other ports
socket
server
Internet address = 138.37.94.248
Internet address = 138.37.88.249

Cliente UDP

Cliente UDP envia uma mensagem ao servidor e recebe resposta

```
import java.net.*;
import java.io.*;
public class UDPClient{
   public static void main(String args[]) {
              // args give message contents and server hostname
              DatagramSocket aSocket = null;
                try {
                  aSocket = new DatagramSocket();
                  byte [] m = args[0].getBytes();
                  InetAddress aHost = InetAddress.getByName(args[1]);
                  int serverPort = 6789;
                  DatagramPacket request = new DatagramPacket(m,  args[0].length(), aHost, serverPort);
                  aSocket.send(request);
                  byte[] buffer = new byte[1000];
                  DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                  aSocket.receive(reply);
                  System.out.println("Reply: " + new String(reply.getData()));
                } catch (SocketException e){System.out.println("Socket: " + e.getMessage());
                } catch (IOException e){System.out.println("IO: " + e.getMessage());
              } finally {if(aSocket != null) aSocket.close();}
   }
}
```

Servidor UDP

Servidor UDP repetidamente recebe uma requisição e retorna ao cliente

```
import java.net.*;
import java.io.*;
public class UDPServer{
        public static void main(String args[]) {
        DatagramSocket aSocket = null;
          try {
                aSocket = new DatagramSocket(6789);
                byte[] buffer = new byte[1000];
                while(true) {
                  DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                 aSocket.receive(request);
                 System.out.println(“Request: " + new String(request.getData()));
                 DatagramPacket reply = new DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(), request.getPort());
                 aSocket.send(reply);
                }
          } catch (SocketException e){System.out.println("Socket: " + e.getMessage());
          } catch (IOException e) {System.out.println("IO: " + e.getMessage());
          } finally {if(aSocket != null) aSocket.close();}
  }
}
```

Cliente TCP

Cliente TCP faz conexão com o server, envia a request e recebe a resposta

```
import java.net.*;
import java.io.*;
public class TCPClient {
        public static void main (String args[]) { // arguments supply message and hostname
of destination
        Socket s = null;
          try {
            int serverPort = 7896;
            s = new Socket(args[1], serverPort);
            DataInputStream in = new DataInputStream( s.getInputStream());
            DataOutputStream out = new DataOutputStream( s.getOutputStream());
            out.writeUTF(args[0]);          // UTF is a string encoding see Sn 4.3
            String data = in.readUTF();
            System.out.println("Received: "+ data) ;
          } catch (UnknownHostException e){
                        System.out.println("Sock:"+e.getMessage());
          } catch (EOFException e){System.out.println("EOF:"+e.getMessage());
          } catch (IOException e){System.out.println("IO:"+e.getMessage());
          } finally {if(s!=null) try {s.close();}catch (IOException
e){System.out.println("close:"+e.getMessage());}}
        }
}
```

Servidor TCP

Servidor TCP faz uma conexão com cada cliente e então ecoa a mensagem cliente

```
import java.net.*;
import java.io.*;
public class TCPServer {
   public static void main (String args[]) {
            try {
                        int serverPort = 7896;
                        ServerSocket listenSocket = new ServerSocket(serverPort);
                        while(true) {
                                    Socket clientSocket = listenSocket.accept();
                                    Connection c = new Connection(clientSocket);
                        }
            } catch(IOException e) {System.out.println("Listen :"+e.getMessage());}
   }
}

// this figure continues on the next slide
```

Servidor TCP (continuação)

```
class Connection extends Thread {
            DataInputStream in;
            DataOutputStream out;
            Socket clientSocket;
            public Connection (Socket aClientSocket) {
               try {
                        clientSocket = aClientSocket;
                        in = new DataInputStream( clientSocket.getInputStream());
                        out =new DataOutputStream( clientSocket.getOutputStream());
                        this.start();
                } catch(IOException e)  {System.out.println("Connection:"+e.getMessage());}
            }
            public void run(){
               try {                                    // an echo server
                        String data = in.readUTF();
                        System.out.println("Received: "+ data) ;
                        out.writeUTF(data);
               } catch(EOFException e) {System.out.println("EOF:"+e.getMessage());
               } catch(IOException e) {System.out.println("IO:"+e.getMessage());
               } finally{ try {clientSocket.close();}catch (IOException e){/*close failed*/}}
            }
}
```

Multicast

Membro Multicast se junta a um grupo, envia e recebe datagramas

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
        public static void main(String args[]){
         // args give message contents & destination multicast group (e.g. "228.5.6.7")
        MulticastSocket s =null;
         try {
                InetAddress group = InetAddress.getByName(args[1]);
                s = new MulticastSocket(6789);
                s.joinGroup(group);
                byte [] m = args[0].getBytes();
                DatagramPacket messageOut =
                        new DatagramPacket(m, m.length, group, 6789);
                s.send(messageOut);


         // continua no próximo slide
```

continuação

```
          // get messages from others in group
                byte[] buffer = new byte[1000];
                for(int i=0; i< 3; i++) {
                   DatagramPacket messageIn =
                        new DatagramPacket(buffer, buffer.length);
                   s.receive(messageIn);
                   System.out.println("Received:" + new String(messageIn.getData()));
                }
                s.leaveGroup(group);
         }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
         }catch (IOException e){System.out.println("IO: " + e.getMessage());
         }finally {if(s != null) s.close();}
   }
}
```

Sockets usado para datagramas

Enviando uma mensagem

```
s = socket(AF_INET, SOCK_DGRAM, 0)
•
•
bind(s, ClientAddress)
•
•
sendto(s, "message", ServerAddress)
```

Recebendo uma mensagem

```
s = socket(AF_INET, SOCK_DGRAM, 0)
•
•
bind(s, ServerAddress)
•
•
amount = recvfrom(s, buffer, from)
```

ServerAddress and *ClientAddress* são endereços socket

Sockets usado para streams

Requisitando uma conexão

```
s = socket(AF_INET, SOCK_STREAM,0)
•
•
connect(s, ServerAddress)
•
•
write(s, "message", length)
```

Ouvindo e aceitando uma conexão

```
s = socket(AF_INET, SOCK_STREAM,0)
•
bind(s, ServerAddress);
listen(s,5);
•
sNew = accept(s, ClientAddress);
•
n = read(sNew, buffer, amount)
```

ServerAddress and *ClientAddress* são endereços socket