COMUNICAÇÃO UTILIZANDO SOCKETS

MCTA025-13 - SISTEMAS DISTRIBUÍDOS

Emilio Francesquini e Fernando Teubl

13 de junho de 2018

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

UFABC

DISCLAIMER

- Estes slides foram preparados para o curso de **Sistemas Distribuídos na UFABC**.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- Estes slides foram adaptados com base no material disponível em
  - `https://docs.oracle.com/javase/tutorial/networking/sockets`.
  - `https://docs.oracle.com/javase/tutorial/networking/datagrams`

SOCKETS

- São o mecanismo básico de comunicação sobre IP
- Tipicamente oferecem três modos de acesso
  - Orientado a conexão
  - Orientado a datagrama
  - Acesso a dados IP de baixo-nível (*raw IP data*)
    - Não disponível diretamente em Java

MODO ORIENTADO A CONEXÃO

- Funciona sobre o protocolo `TCP/IP`
- Oferece a garantia de entrega e ordem de entrega dos pacotes
- Facilitam a implementação pois abstrações de linguagens como *streams* podem ser utilizados
- Dá suporte à troca de dados bidirecional, mas pode ser configurado para ser unidirecional
  - Feito em Java pelos métodos `shutdownInput` e `shutdownOutput`
- Tem um maior *overhead* (devido as garantias oferecidas)

MODO ORIENTADO A DATAGRAMAS

- Funciona sobre o protocolo **UDP/IP**
- Trabalha com a política do **melhor esforço** (*best effort*)
  - Não garante a entrega ou a ordem
- Cada mensagem é um datagrama
  - Tupla (remetente, destinatário, conteúdo)
- Por não oferecer as garantias oferecidas por TCP/IP, tem menos *overhead* e consequentemente é mais rápido que o modo orientado a conexão.

SOCKETS - ORIENTADO A CONEXÃO EM JAVA

Um *Socket* é a representação de cada uma das extremidades de um link de comunicação através de uma rede

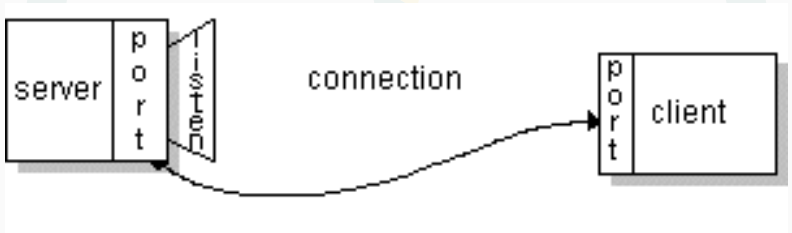
- Em Java há duas classes diferentes que representam sockets (ambas em `java.net`):
  - `Socket` - Representa o lado do cliente (que inicia a conexão)
  - `ServerSocket` - Representa o lado do servidor (aguarda os clientes conectarem)

SOCKETS - INICIANDO UMA CONEXÃO

- Servidor
  - Tipicamente, é executado em um computador específico com um endereço e porta de conexão conhecidos pelos clientes
    - Diferentes servidores em uma mesma máquina utilizam diferentes portas
  - Vincula-se (*binds*) à uma porta específica e fica aguardando (ouvindo, *listening*) por conexões
- Cliente
  - Envia um pedido de conexão ao servidor utilizando o seu endereço e a porta (que são conhecidos de antemão)
  - Durante este processo, para que seja possível estabelecer uma comunicação de 2 vias, também escolhe uma porta na qual ele se vincula (o número desta porta é tipicamente escolhido pelo próprio sistema)

SOCKETS - ESTABELECENDO UMA CONEXÃO

- Servidor
  - Aceita a conexão e recebe um novo socket
    - Vinculado à mesma porta local
    - Endereço remoto vinculado ao endereço/porta do cliente
    - O socket original continua ativo ouvindo por conexões
- Cliente
  - Caso a conexão seja aceita pelo servidor a conexão é efetuada com sucesso e pode-se iniciar a comunicação



ENDPOINTS

- Ambos, cliente e servidor, podem agora ler e escrever diretamente do socket. A **diferenciação entre cliente e servidor passa a depender exclusivamente da aplicação**
- Um *endpoint* é o par de um endereço IP e um número de porta. Cada uma das conexões TCP pode ser identificada unicamente pelos seus dois endponints.
- Como cada conexão recebe um número de porta diferente, é possível estabelecer múltiplas conexões entre o cliente e o servidor.

SOCKETS - EXEMPLO ECHO SERVER

Network Working Group
Request for Comments: 862

J. Postel
ISI
May 1983

Echo Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement an Echo Protocol are expected to adopt and implement this standard.

A very useful debugging and measurement tool is an echo service. An echo service simply sends back to the originating source any data it receives.

TCP Based Echo Service

One echo service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 7. Once a connection is established any data received is sent back. This continues until the calling user terminates the connection.

UDP Based Echo Service

Another echo service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 7. When a datagram is received, the data from it is sent back in an answering datagram.

ECHO CLIENT

```
try (
    Socket echoSocket = new Socket(hostName, portNumber);
    PrintWriter out =
        new PrintWriter(echoSocket.getOutputStream(), true);
    BufferedReader in =
        new BufferedReader(
            new InputStreamReader(echoSocket.getInputStream()));
    BufferedReader stdIn =
        new BufferedReader(new InputStreamReader(System.in))
) {
    String userInput;
    while ((userInput = stdIn.readLine()) != null) {
        out.println(userInput);
        System.out.println("echo: " + in.readLine());
    }
} catch (UnknownHostException e) {
    System.err.println("Don't know about host " + hostName);
} catch (IOException e) {
    System.err.println("Couldn't get I/O for the connection to " +
        hostName);
}
```

ECHO SERVER

```
try (
    ServerSocket serverSocket = new ServerSocket(port);
    Socket clientSocket = serverSocket.accept();
    PrintWriter out =
        new PrintWriter(clientSocket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(
        new InputStreamReader(clientSocket.getInputStream()));
) {
    String inputLine;
    while ((inputLine = in.readLine()) != null) {
        out.println(inputLine);
    }
} catch (IOException e) {
    System.out.println("Exception trying to listen on port "
        + portNumber + " or listening for a connection");
    System.out.println(e.getMessage());
}
```

O LADO DO SERVIDOR

- Utilizando `java.net.ServerSocket` o servidor pode aceitar uma conexão de um cliente
  - O método `accept` suspende a execução até que uma requisição de cliente tenha sido recebida
  - Um socket é criado **na mesma porta** para controlar a conexão com o novo cliente
  - É possível (e desejável) ter um servidor que dê suporte a múltiplos clientes simultaneamente

```
while (true) {
  //aceita a conexão
  //cria um thread para lidar com o cliente
}
```

IMPLEMENTANDO OUTROS PROTOCOLOS

1. Estabeleça a conexão e obtenha um socket
2. Obtenha os streams de entrada e saída do socket
3. Leia e escreva no socket obedecendo os protocolo do servidor
4. Feche os streams
5. Feche o socket

EXERCÍCIO 1

- Crie um protocolo para comunicação entre dois usuários através da rede usando TCP/IP
- Ambos devem ser capazes de ler e escrever mensagens um para o outro
- A comunicação entre eles termina (e ambos os programas devem sair) quando qualquer um dos participantes escrever uma linha contendo apenas a palavra `SAIR`
- Mensagens são enviadas linha a linha

EXERCÍCIO 2

- Altere o seu Exercício 1 para que ele aceite múltiplos clientes.
- Seu programa agora deverá implementar uma conversa em grupo.
- Mensagens enviadas para o servidor devem ser mostradas em todos os clientes conectados.
- Um cliente pode sair do grupo da mesma maneira que antes (escrevendo a linha `SAIR`), contudo o servidor deverá permanecer no ar ainda que o último cliente se desconecte

Lado do remetente

```
DatagramSocket clientSocket = new DatagramSocket ();
InetAddress addr =
    InetAddress.getByName ("ufabc.edu.br");
String mensagem = "Vai cair na prova?";
byte[] buffer = mensagem.getBytes();
DatagramPacket datagrama = new DatagramPacket (
    buffer, buffer.length, addr, 1234);
clientSocket.send (datagrama);
```

Lado do remetente - Recebendo a resposta

```
DatagramPacket resposta = new
    DatagramPacket (new byte[512], 512);
clientSocket.receive (resposta);
System.out.println (resposta.getData()
  + "\n" + resposta.getLength()
  + "\n" + resposta.getAddress()
  + "\n" + resposta.getPort());

//Quando não houver mais comunicações a se fazer
clientSocket.close();
```

Lado do destinatário

```
DatagramSocket serverSocket =
    new DatagramSocket (1234);
DatagramPacket mensagem = new DatagramPacket (
    new byte[512], 512);
serverSocket.receive (mensagem);

String resposta = "Provavelmente!";
byte[] buffer = resposta.getBytes();
DatagramPacket datagramaResposta = new DatagramPacket (
    buffer, buffer.length,
    question.getAddress(), question.getPort());
serverSocket.send (datagramaResposta) ;

//Quando a comunicação acabar
serverSocket.close();
```

EXERCÍCIO 3

- Altere novamente o seu Exercício para que passe a utilizar UDP/IP em vez de TCP/IP