Sistemas Distribuídos

2018/2019

Aula 1

- Descoberta de serviços

Porquê "Descobrir Serviços?"

A **descoberta de serviços** visa que o acesso a um serviço (local) possa fazer-se sem informação prévia, tal como: `<ip:port>` ou URI/URL.

Uma das motivações para usar a descoberta é permitir que um sistema complexo se auto-organize e auto-configure, com um mínimo de informação estática inicial.

Também permite que as componentes possam mudar de localização (endereço) de forma mais transparente.

A descoberta de serviços pode ser facilmente implementada em redes locais que suportem *IP Multicast*.

No código abaixo, o utilizador de um cliente HTTP precisa de, manualmente, fornecer a localização do servidor HTTP antes de poder realizar um pedido, excepto se esta for fixa e pré-determinada.

Um mecanismo de descoberta poderá fornecer essa informação automaticamente, em tempo execução.

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

public class HttpClient {

    public static void main(String[] args) throws IOException {

        String serverUrl = args.length > 0 ? args[0] : "http://localhost:8888/";

        String objectUrl = serverUrl + "earth.jpg";

        URLConnection conn = new URL(objectUrl).openConnection();

        conn.getHeaderFields().forEach( (header,value) -> {
            System.out.printf("%s: %s\n", header, value);
        });
        conn.getContent();
    }
}
```

Como implementar a Descoberta de Serviços?

Usando **IP Multicast** temos duas opções:

- descoberta iniciada pelo **servidor**;
- descoberta iniciada pelo **cliente**.

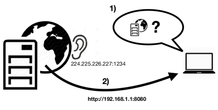
Descoberta iniciada pelo servidor

Os serviços anunciam periodicamente a sua presença, enviando o seu URI/URL para um grupo IP multicast e porto pré-acordados.

Descoberta iniciada pelo cliente

O cliente interessado num serviço envia uma interrogação para um grupo IP Multicast e porto pré-acordados, contendo o nome do serviço que deseja.

O(s) servidor(es) responderão ao cliente com o seu URI/URL (diretamente, em modo *unicast*).



Fiabilidade

Não sendo a comunicação IP Multicast fiável, em ambas as modalidades de descoberta, é necessário atender a este problema.

Na descoberta iniciada pelo servidor, o cliente precisa de escutar o grupo multicast tempo suficiente até poder concluir que o serviço (provavelmente) não se encontra disponível, ou até encontrar todas as instâncias do serviço que existem.

Na descoberta iniciada pelo cliente, a interrogação precisa de ser repetida várias vezes para haver várias hipóteses de esta ser ouvida pelo(s) servidor(es) e da resposta deste(s) chegar ao cliente.

Comunicação Multiponto (Revisão)

IP Multicast (Recepção)

O excerto de código seguinte ilustra como receber mensagens em modo *multicast*/multiponto de um grupo *multicast*.

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

final int PORT = 1234;
final int MAX_DATAGRAM_SIZE = 65536;
final InetAddress group = InetAddress.getByName( args[0] ) ;
if( ! group.isMulticastAddress()) {
    System.out.println( "Not a multicast address (use range : 224.0.0.0 -- 239.255.255.255)"
);
    System.exit( 1);
}

try( MulticastSocket socket = new MulticastSocket( PORT )) {
    socket.joinGroup( group);
    while( true ) {
        byte[] buffer = new byte[MAX_DATAGRAM_SIZE] ;
        DatagramPacket request = new DatagramPacket( buffer, buffer.length ) ;
        socket.receive( request );
        System.out.write( request.getData(), 0, request.getLength() ) ;
    }
}
```

Recordar que é usada uma gama de endereços IP dedicada; que o receptor utiliza uma classe de *sockets* própria para receber *multicasts*; e, que precisa de se juntar ao grupo *multicast* para poder receber mensagens.

IP Multicast (Envio)

O excerto de código seguinte ilustra como enviar mensagens endereçadas um grupo *multicast*.

```
final int port = 9000 ;
final InetAddress group = InetAddress.getByName( args[0] ) ;

if( ! group.isMulticastAddress()) {
    System.out.println( "Not a multicast address (use range : 224.0.0.0 -- 239.255.255.255)"
);
}

byte[] data = "hello?".getBytes();
try(DatagramSocket socket = new DatagramSocket()) {
    DatagramPacket request = new DatagramPacket( data, data.length, group, port ) ;
    socket.send( request );
}
```

Recordar que basta endereçar as mensagens ao grupo, não sendo necessário usar nenhum socket especial ou pertencer ao grupo se apenas se deseja enviar para um endereço destino *multicast*.

Aspetos práticos

Em alguns sistemas poderá ser necessário forçar que a comunicação multicast utilize a pilha IPv4 e não IPv6.

Para tal, existem duas opções:

- Programaticamente, no código do cliente e servidor, incluir no arranque:
  `System.setProperty("java.net.preferIPv4Stack", "true");`
- Na invocação dos programas:
  `java -Djava.net.preferIPv4Stack=true ...`

Em sistemas *multihome*, isto é, dispondo de várias interfaces de rede, poderá ser necessário que a operação `joinGroup` sobre o MulticastSocket seja feita em todas as interfaces, ou especificamente na interface da rede de onde se espera receber a comunicação multiponto.

Exercício

O trabalho prático tem como um dos requisitos a auto-configuração do sistema a desenvolver. Será, então, necessário que as componentes se encontrem automaticamente, sem depender de ficheiros de configuração com endereços fixos, ou nomes de máquinas pré-definidos.

Para o efeito, o trabalho deverá incluir um mecanismo de descoberta baseado no envio de anúncios periódicos pelos servidores, ou seja, implementando a modalidade de descoberta iniciada pelos servidores.

O projeto P1-Descoberta/SD2019-Labs-P1 incluído no repositório contém o esboço de uma classe Java que implementa parte da descoberta de serviços.

Analise o código da classe discovery.Discovery e complete a sua implementação. Confirme que a sua solução está correta usando o código do pacote http também contido no projeto.