Sockets

Programação para dispositivos móveis

Prof. Allan Rodrigo Leite

Sockets

- São abstrações das camadas de rede para aplicações que necessitam se comunicar com outras aplicações
  - Mecanismo básico de comunicação sobre IP
  - Utilizado para comunicação entre processos por meio de uma rede

- Fornece três modos de acesso
  - Modo orientado a conexão (connection-oriented)
  - Modo orientado a datagrama (datagram-oriented)
  - Acesso a dados IP de baixo nível (raw ip data)

Modo orientado a conexão

- Funciona sobre o protocolo TCP (Transmission Control Protocol)
  - Na maioria dos casos são serviços confiáveis
  - Sem perdas de dados na rede
  - Garantia de ordem dos pacotes
  - Podem ser utilizados para fluxo de dados (data stream)

- Desvantagens
  - É mais lento que o modo orientado a datagrama
  - Comportamento servidor diferente do comportamento cliente

Modo orientado a datagrama

- Funciona sobre o protocolo UDP (User Datagram Protocol)
  - Protocolo de baixa latência e tolerante a perda de dados
  - Não garante a ordem de recebimento dos dados
  - Comumente usado em aplicações sensíveis à latência
    - Stream de vídeo ou áudio
    - Jogos on-line

- Desvantagens
  - Não garante uma comunicação confiável
  - Não fornece serviço de controle de congestionamento

Gerenciamento de sockets

- Pacote `java.net`
  - Fornece classes para comunicação em rede
    - Modo orientado a conexão: `Socket` e `ServerSocket`
    - Modo orientado a datagrama: `DatagramSocket`, `MulticastSocket`
    - Acesso a dados IP de baixo nível: `SocketImpl`

  - Permite a realização de operações básicas em um socket
    - Iniciar um servidor
    - Iniciar uma conexão
    - Enviar e receber dados (fluxos de entrada e saída)

Iniciando um servidor

- Para iniciar um servidor deve ser especificada a porta a ser utilizada
  - A porta pode variar entre `1` e `65535`
  - Porém, normalmente utiliza-se portas acima de `1000`
  - Não é permitido mais de um servidor utilizando a mesma porta
    - Considerando execução dos servidores no mesmo sistema operacional
  - Caso ocorrer algum erro ao iniciar um servidor socket na porta especificada, uma exceção do tipo `IOException` será lançada

- A classe `ServerSocket` possui a implementação de um servidor
  - A classe `Socket` permite o envio e recebimento de uma conexão socket

Iniciando um servidor

```
public class Servidor {
  public static void main(String args[]) throws IOException {
    var servidor = new ServerSocket(12345);
    System.out.println("Servidor iniciado na 12345!");

    var socket = servidor.accept();
    System.out.println("Conexão estabelecida!");

    //...
  }
}
```

Iniciando um cliente

```
public class Cliente {
  public static void main(String args[]) throws IOException {
    var servidor = new Socket("127.0.0.1", 12345);
    System.out.println("Cliente conectado!");

    //...
  }
}
```

Manipulando dados de um socket

- A classe Socket possui dois métodos para manipular as streams de entrada e saída
  - `socket.getInputStream()`
  - `socket.getOutputStream()`

- Um stream é um fluxo de dados
  - Possui uma direção (input ou output)
    - Input: recebe dados da outra aplicação
    - Output: envia dados para a outra aplicação
  - Suporta qualquer tipo de dados (texto, áudio, vídeo, arquivo, etc.)
    - Todo stream precisa ser fechada através do método `close()`
    - Assim que a conexão não for mais necessária

Manipulando dados de um socket

```
public class Cliente {
  public static void main(String[] args) throws IOException {
    var cliente = new Socket("127.0.0.1", 12345);
    var teclado = new Scanner(System.in);
    var saida = new PrintStream(cliente.getOutputStream());

    while (teclado.hasNextLine()) {
      saida.println(teclado.nextLine());
    }

    saida.close();
    teclado.close();
    cliente.close();
  }
}
```

Manipulando dados de um socket

```
public class Servidor {
  public static void main(String[] args) throws IOException {
    var servidor = new ServerSocket(12345);
    System.out.println("Servidor iniciado na porta 12345!");

    var cliente = servidor.accept();
    System.out.println("Cliente " + cliente.getInetAddress().getHostAddress());
    var saida = new Scanner(cliente.getInputStream());

    while (saida.hasNextLine()) {
      System.out.println(saida.nextLine());
    }

    saida.close();
    servidor.close();
    cliente.close();
  }
}
```

Manipulando múltiplos sockets em um servidor

- É possível fazer com que um servidor manipule simultaneamente múltiplos sockets
  - Utilizando técnicas de programação multi-thread

- Neste caso, a cada nova conexão estabelecida pode ser iniciada uma thread para gerenciar o fluxo de dados do socket
  - Além disso, é possível que um socket possua múltiplas threads manipulando seu próprio fluxo de dados
  - Neste caso, geralmente é utilizada duas threads
    - Uma para manipular o fluxo de entrada
    - Outra para manipular o fluxo de saída

Manipulando múltiplos sockets em um servidor

```
public class Conexao extends Thread {
  private Socket socket;
  public Conexao(final Socket socket) { this.socket = socket; }

  public void run() {
    var saida = new Scanner(cliente.getInputStream());
    while (saida.hasNextLine()) { System.out.println(saida.nextLine()); }
  }

  public static void main(String args[]) throws IOException {
    var servidor = new ServerSocket(12345);
    System.out.println("Servidor iniciado na 12345!");

    while (true) {
      var socket = servidor.accept();
      System.out.println("Conexão estabelecida!");
      new Conexao(socket).start();
    }
  }
}
```

Exercícios

- Implemente um aplicativo de conversas instantâneas utilizando sockets. Este aplicativo deve ter um programa que implementa um servidor e um programa que implementa o cliente. Os seguintes requisitos precisam ser atendidos
  - Tanto o cliente quando o servidor devem ser capazes de enviar e receber dados
  - Os dados recebidos devem ser exibidos na saída padrão (`System.out`)
  - Os dados enviados devem ser capturados da entrada padrão (`System.in`)
  - A qualquer momento o cliente pode finalizar a comunicação ao informar o comando `!sair`
  - O servidor deve manter um log em arquivo dos clientes que se conectaram, contendo os endereços IP e a data e hora de conexão

Sockets

Programação para dispositivos móveis

Prof. Allan Rodrigo Leite