



Java Standard Edition (JSE)

15. Sockets



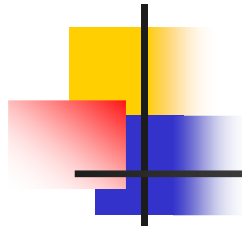
Esp. Márcio Palheta

Gtalk: marcio.palheta@gmail.com



Agenda

- Protocolo de comunicação;
- Porta de acesso;
- Socket;
- Servidor de transações;
- Cliente;
- Exercícios;



A API e os conceitos

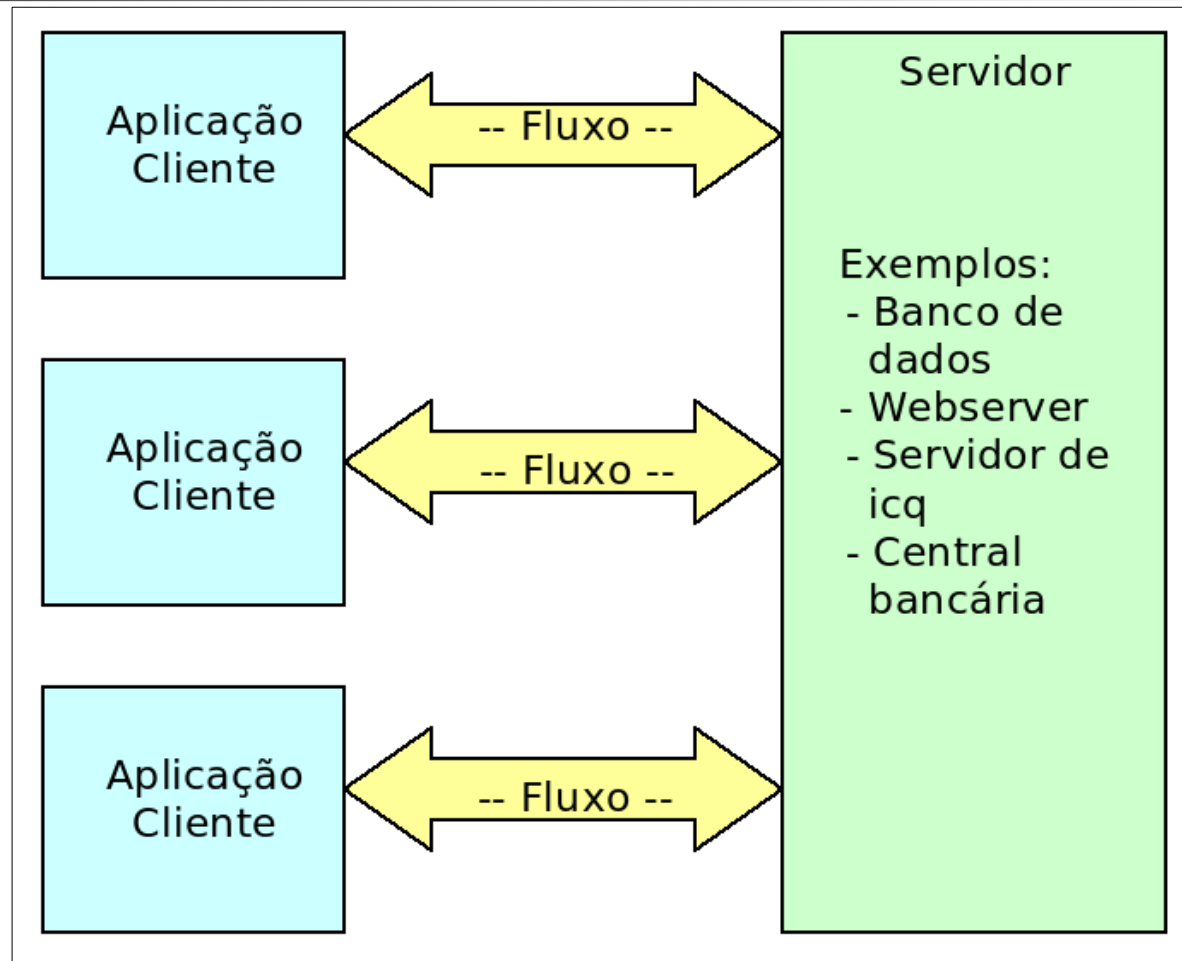
- Neste capítulo, estudaremos a API de Sockets do pacote java.net;
- Perceba que estamos colocando em prática todos os conceitos a cerca bibliotecas e interfaces, que aprendemos nesse curso;
- Fica mais fácil avançar na API, uma vez que a base está sólida;



Protocolos

- A fim de garantir a comunicação entre estações remotamente dispostas, definiu-se conjuntos de regras, ditos Protocolos;
- O protocolo que usaremos é o Transmission Control Protocol – TCP;
- Como o TCP, podemos estabelecer um fluxo de dados entre estações distintas;

Fluxo de dados TCP





Sobre a conexão

- Podemos conectar mais de um cliente a um servidor;
- Os clientes podem trocar mensagens usando o servidor;
- Para trabalhar com comunicação de dados, o java oferece as classes do pacote [java.net](#);
- O TCP vai garantir a entrega dos nossos pacotes de dados;



Porta de acesso

- É comum que várias aplicações locais troquem dados com outras do Servidor;
- Mas temos apenas uma conexão física;
- Assim como há **IP** para identificar CPU, a **porta** é a solução para acesso a aplicações de uma máquina;
- A **porta** varia de 0 a 65535. Só é possível conexão se uma porta estiver liberada;



Socket

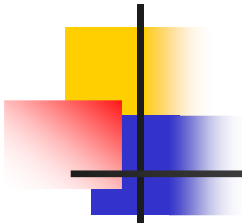
- Digamos que um cliente se conecta a um programa rodando na porta **80(http)**;
- Precisamos esperar que ele desconecte para conectarmos outro cliente?
- Após aceitar a conexão, o **servidor** redireciona o cliente para outra porta;
- Dessa forma, a porta 80 fica liberada, aguardando outra conexão;
- Em java, usamos **threads** para a troca;



Aplicação servidora

- Aceita conexão e dados de aplicações clientes;
- O servidor deve abrir uma porta e ficar escutando por uma conexão cliente;
- Após a conexão, o servidor aceita dados (**Streams**) enviados pelo cliente;
- Por fim, é necessário fechar as conexões;

Exercício 01 – Servidor.java



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

    String mensagem = "";

    1 // Criando servidor
    ServerSocket server = new ServerSocket(12345);
    System.out.println("SERVER: Abertura da porta 12345");

    2 // Aguardando conexões
    Socket connection = server.accept();
    System.out.println("SERVER: Conectado - "
        + connection.getInetAddress().getHostName());

    3 // Configuração de objetos de RW
    ObjectInputStream entrada = new ObjectInputStream(connection
        .getInputStream());

    //Processando conexão
    do {
        4 //Leitura do objeto
        mensagem = (String) entrada.readObject();
        System.out.println(mensagem);
    } while (!mensagem.equals("CLIENTE: TERMINATE"));

    5 //Fechando as conexões
    entrada.close();
    server.close();
    System.out.println("SERVIDOR ENCERRADO");
}
```

Exercício 02 – Cliente.java

```
public static void main(String[] args) throws IOException {  
    // Criando conexão  
    1 Socket connection = new Socket("127.0.0.1", 12345);  
    System.out.println("CLIENT: Conectado à porta 12345");  
  
    // Configuracao do objeto de envio de mensagem  
    2 ObjectOutputStream saida = new ObjectOutputStream(connection  
        .getOutputStream());  
  
    // Envio de mensagens  
    String mensagem = "";  
    do {  
        mensagem = "CLIENTE: " + JOptionPane.showInputDialog("Mensagem");  
        //Escrita do objeto String  
        3 saida.writeObject(mensagem);  
        saida.flush();  
    } while (!mensagem.equals("CLIENTE: TERMINATE"));  
  
    // Encerrando conexão  
    4 saida.close();  
    connection.close();  
    System.out.println("CLIENTE ENCERRADO");  
}
```



Considerações

- O que aconteceu nos exercícios anteriores?
- Foram criados um **Servidor** e um **Cliente**
- O cliente envia objetos String para o servidor;
- O servidor interpreta(imprime) os objetos recebidos;
- A classe String implementa a interface Serializable;



Exercício 03

- Crie a classe Mensagem.java

```
public class Mensagem implements Serializable{  
    private String assunto;  
    private String mensagem;  
  
    public Mensagem(String assunto, String mensagem) {  
        this.assunto = assunto;  
        this.mensagem = mensagem;  
    }  
  
    public String toString() {  
        return "Assunto: "+assunto+  
            "\nMensagem: "+mensagem;  
    }  
    public String getAssunto() {  
    }  
    public void setAssunto(String assunto) {  
    }  
    public String getMensagem() {  
    }  
    public void setMensagem(String mensagem) {  
    }  
}
```



Exercício 04

- Atualize o código das classes Cliente e Servidor, para implementar a troca de objetos **Mensagem**, ao invés de objetos **String**;



Bibliografia

- Java - Como programar, de Harvey M. Deitel
- Use a cabeça! - Java, de Bert Bates e Kathy Sierra
- (Avançado) Effective Java Programming Language Guide, de Josh Bloch



Referências WEB

- SUN: www.java.sun.com
- Threads:
<http://download.oracle.com/javase/tutorial/essential/concurrency/>

Fóruns e listas:

- Javaranch: www.javaranch.com
- G.U.J: www.guj.com.br

Apostilas:

- Argonavis: www.argonavis.com.br
- Caelum: www.caelum.com.br



Java Standard Edition (JSE)

15. Sockets



Esp. Márcio Palheta

Gtalk: marcio.palheta@gmail.com