

Programação Java para Web

Projeto UERJ-FAPERJ

Prof. Austeclynio Pereira
2023

Bibliografia

- Java para Web com Servlets, JSP e EJB – Budi Kurniawan
- Murach's Java Servlets and JSP – Andrea Steelman
- Head First Servlets & JSP – Bryan Basham, Kathy Sierra e Bert Bates
- ◆ Enterprise Java Developer's Guide – S. Narayanan, Junhe Liu
- ◆ The J2EE Tutorial – Sun Microsystems
- ◆ Core Servlets and JavaServer Pages – Vol I – Marty Hall
- ◆ Como o Tomcat Funciona – Budi Kurniawan e Paul Deck

Programação Java para Web

- ♦ Início em 28/02/2023;
- ♦ Término em 11/04/2023;
- ♦ Horário: das 9h às 12h;
- ♦ Dias da semana: 3as e 5as;
- ♦ Conteúdo do curso:
Será enviado para o e-mail dos participantes

Avaliação - Projeto ao final do curso;

Foco do Curso

- ◆ Capacitar para desenvolver aplicações back-end em Java;
- ◆ Apresentando:
 - Fundamentos da arquitetura cliente servidor.
 - Padrão MVC.
 - Desenvolvimento de servlets.
 - Autenticação.
 - Gerenciamento de sessões.
 - Conexão com o banco de dados MySql.
 - Ajax Json.

Ferramentas para o desenvolvimento dos sites

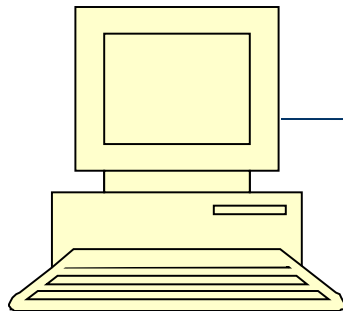
- ◆ IDE NetBeans ou Eclipse ou JCreator.
- ◆ MySql a partir da versão 5.7.
- ◆ TomCat a partir da versão 7.
- ◆ Java a partir da versão 1.8.0_91.

Modelo cliente-servidor

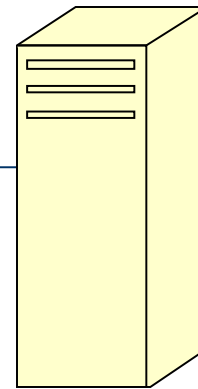
- ◆ Principal padrão utilizado na Internet.
- ◆ Os clientes requisitam os serviços e o servidor realiza os serviços solicitados pelos clientes.
- ◆ Necessidade de uma rede de computadores, de um protocolo de comunicação e de um mecanismo de localização.

Modelos Arquiteturais – uma camada

Máquina Cliente



Servidor Web

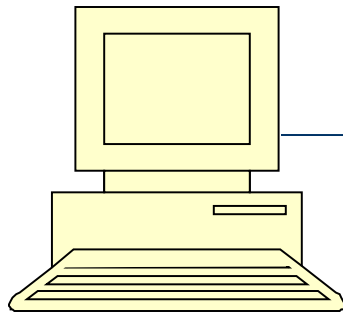


Web Browser

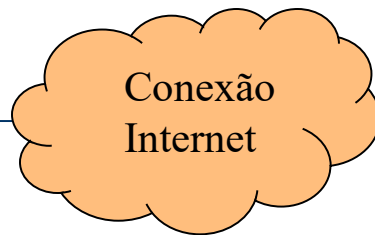
- **Apresentação**
- **Regras do negócio**
- **Persistência**

Modelos Arquiteturais – duas camadas

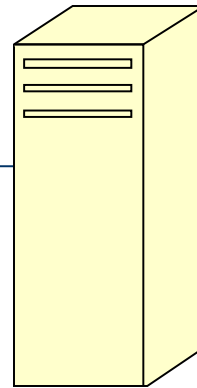
Máquina Cliente



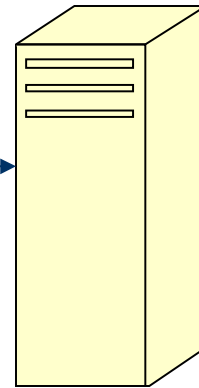
Web Browser



Servidor Web



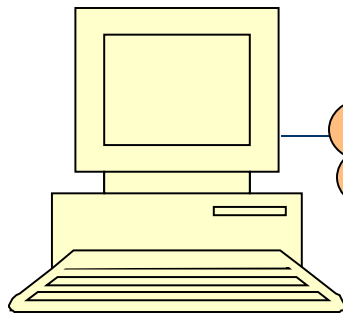
Servidor DataBase



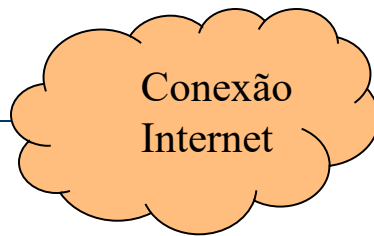
- **Apresentação**
- **Regras do negócio**

Modelos Arquiteturais – três camadas

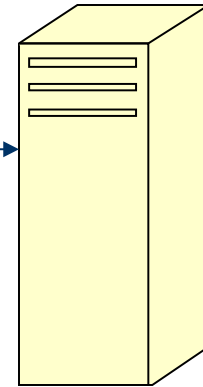
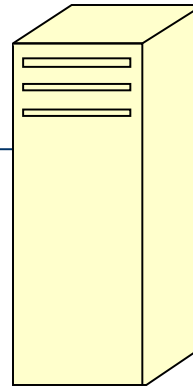
Máquina Cliente



Web Browser



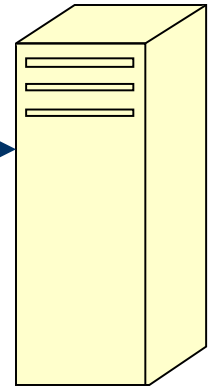
Servidor Web



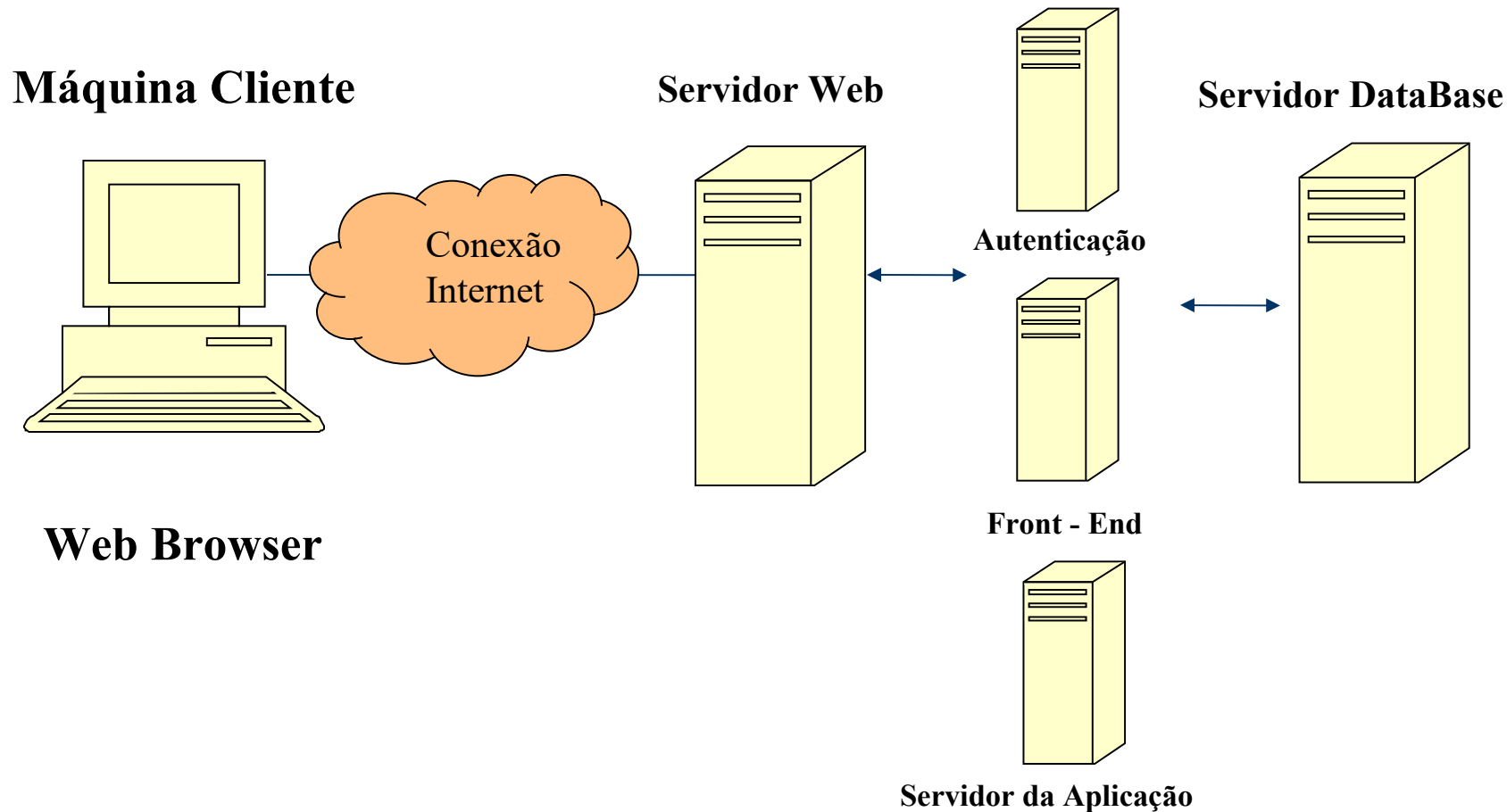
Servidor de Aplicações



Servidor DataBase



Modelos Arquiteturais – três camadas



Ciclo Request Response

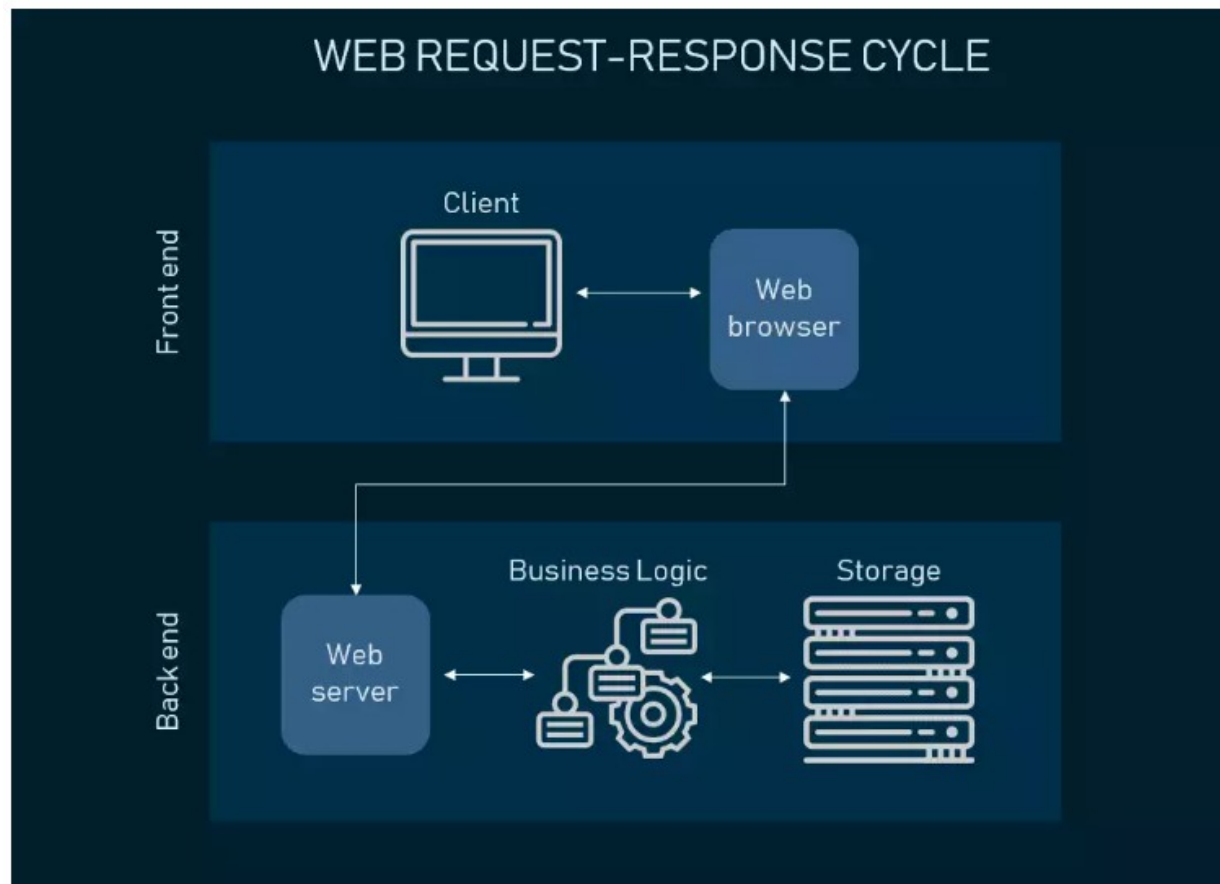
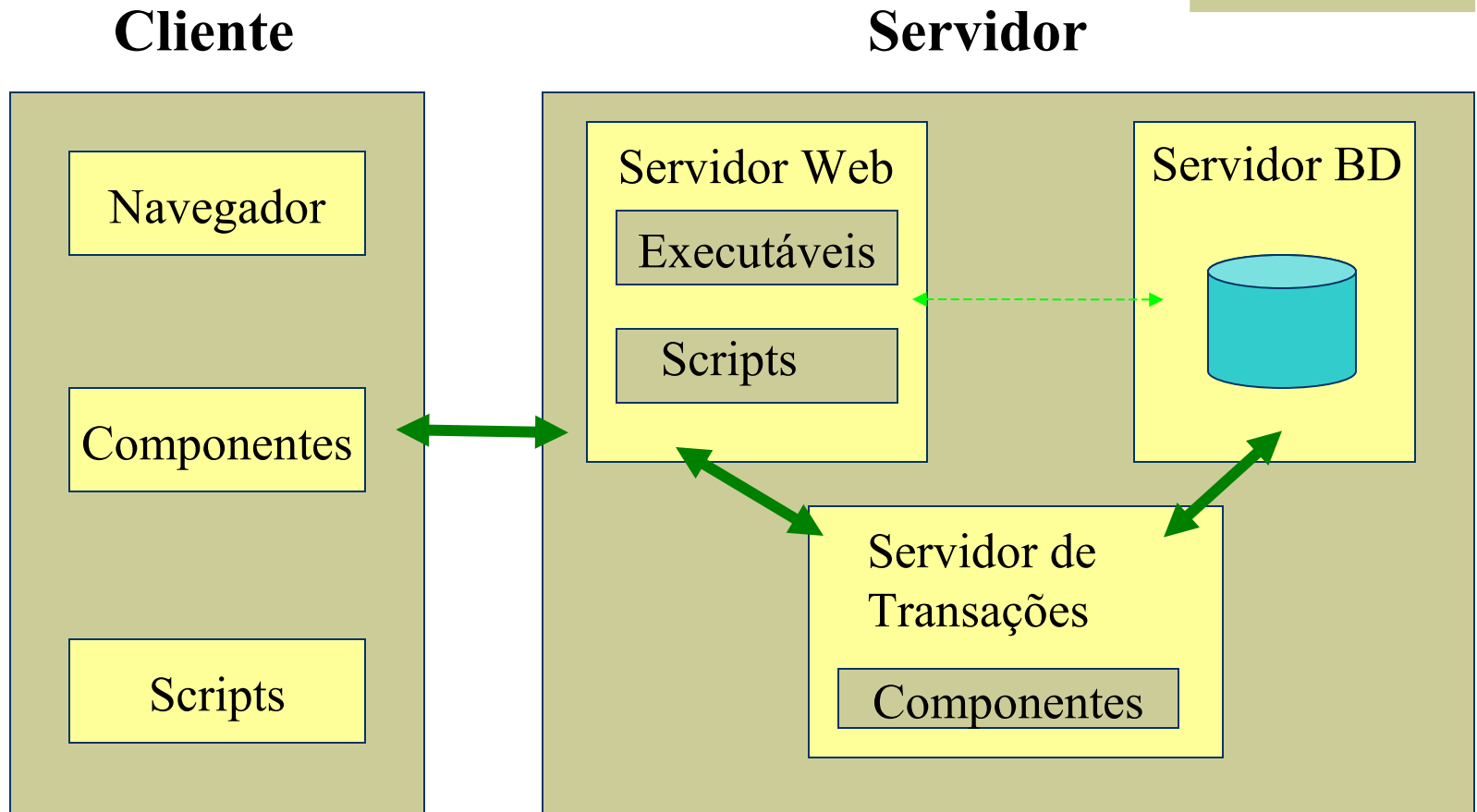


Figura extraída de <https://www.altexsoft.com/>

Elementos típicos de uma aplicação *web*



Elementos típicos de uma aplicação *web* - lado cliente

- ♦ **Scripts** – normalmente utilizados para validar dados de entrada. Diminui o número de requisições ao servidor. Ex: JavaScript.
- ♦ **Componentes** – podem conter parte da lógica do negócio, desonerando o servidor. Exs: Applets e Active-X.

Elementos típicos de uma aplicação *web* - lado servidor

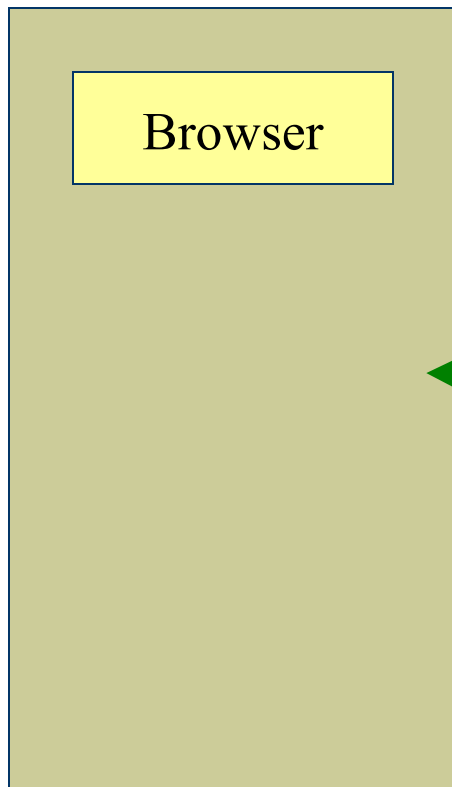
- ♦ **Common Gateway Interface** – módulo executável que produz páginas e informações para o cliente. Cada invocação gera um outro processo.
- ♦ **Scripts** – gera uma página HTML para o cliente ou transfere a página para outro servidor. Podem misturar lógica do negócio com apresentação. Exs.: ASP, JSP, PHP, Angular e React.
- ♦ **Componentes** – módulos executáveis invocados por scripts ou por outros módulos executáveis. Exs.: COM+ e EJB.
- ♦ **Executáveis** – executados em um mesmo processo, capacidade de gerenciar sessões, formulários e cookies.Ex.: Servlets.

Estilos Arquiteturais

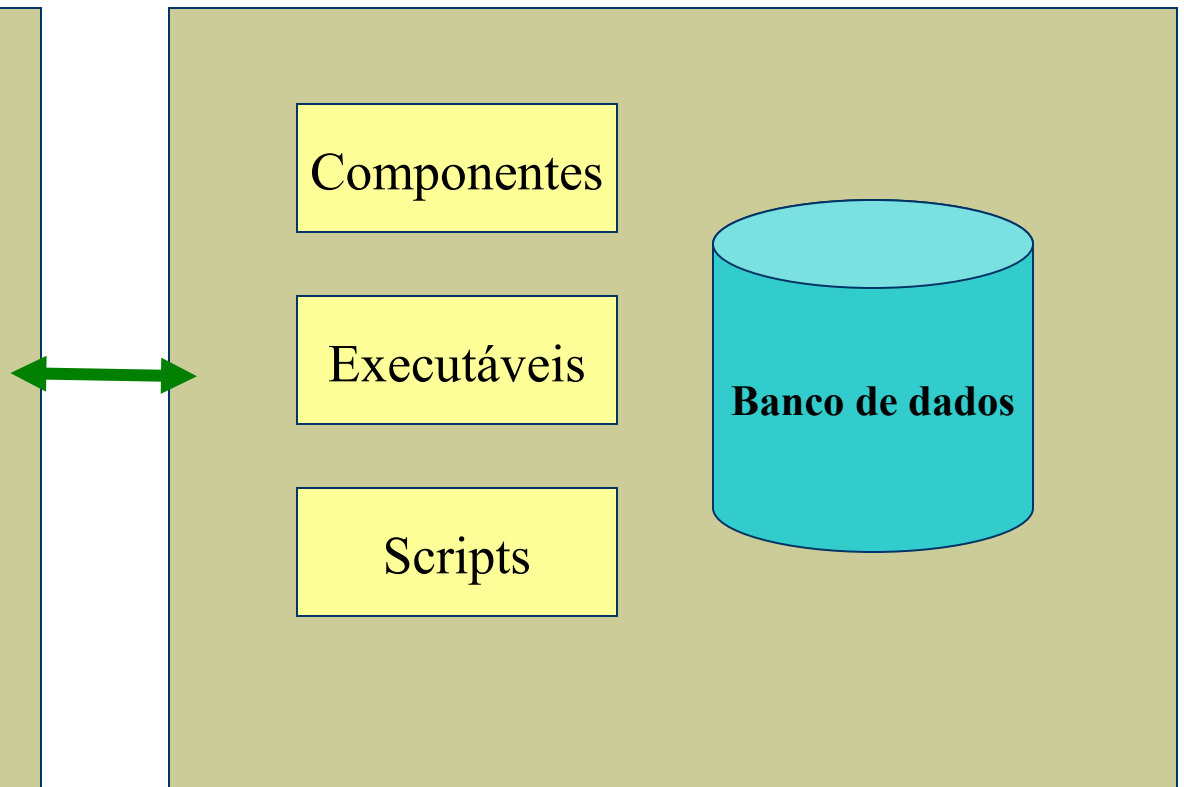
- ♦ **Thin client** – utilização mínima dos recursos da máquina cliente, praticamente tudo é tratado pelo servidor.
- ♦ **Scripted client** – *scripts* na máquina cliente para a verificação de dados.
- ♦ **Thick client** – distribuição da lógica do negócio entre a máquina cliente e a máquina servidora.

Elementos típicos da Internet thin client

Cliente

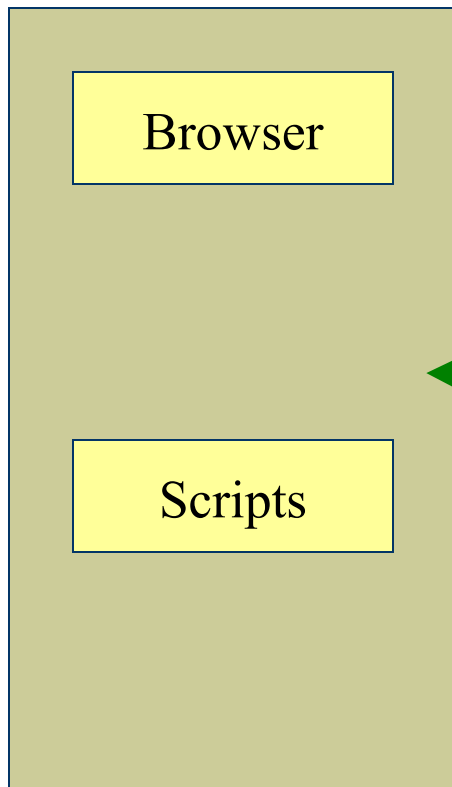


Servidor

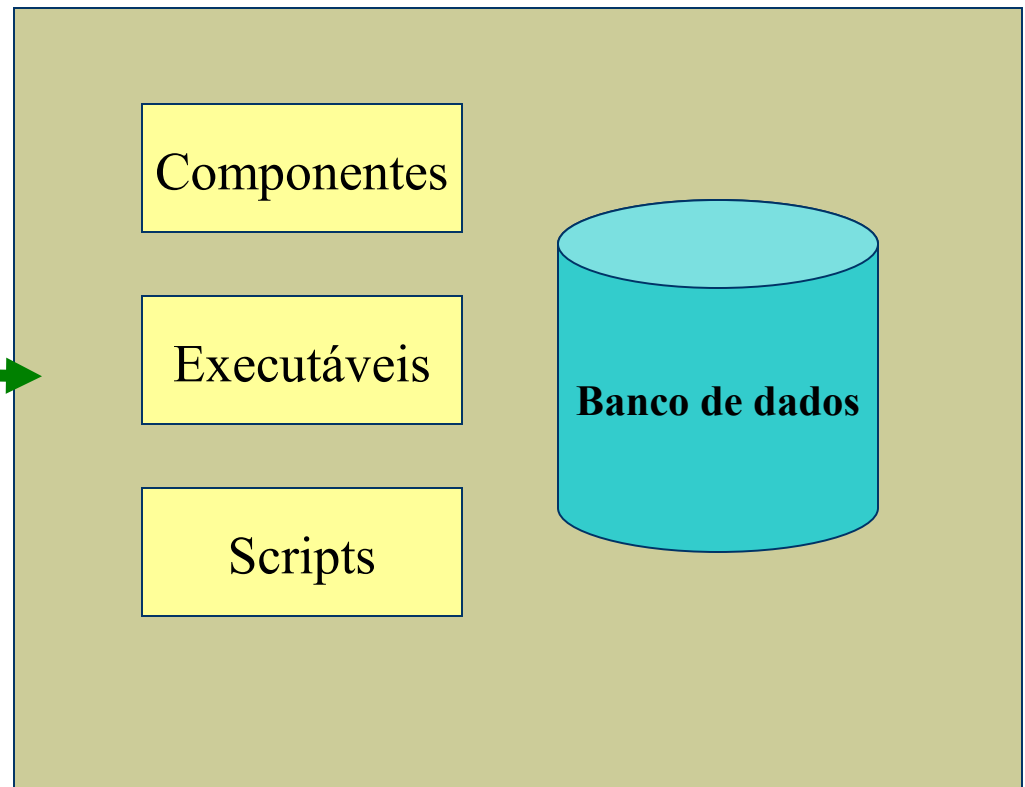


Elementos típicos da Internet scripted client

Cliente

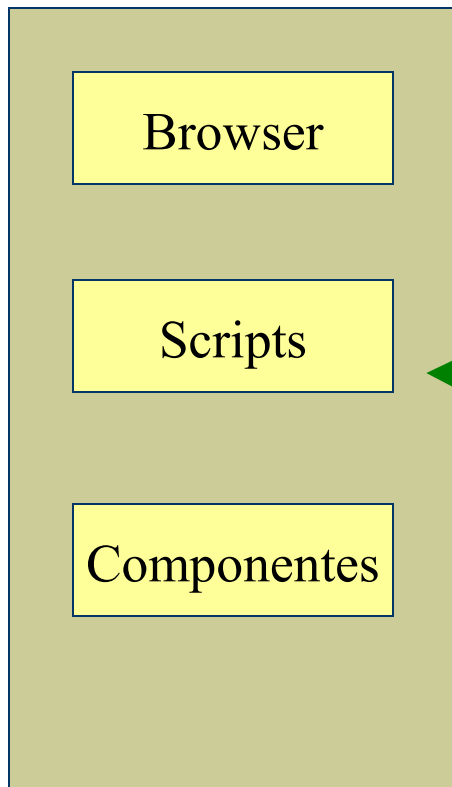


Servidor

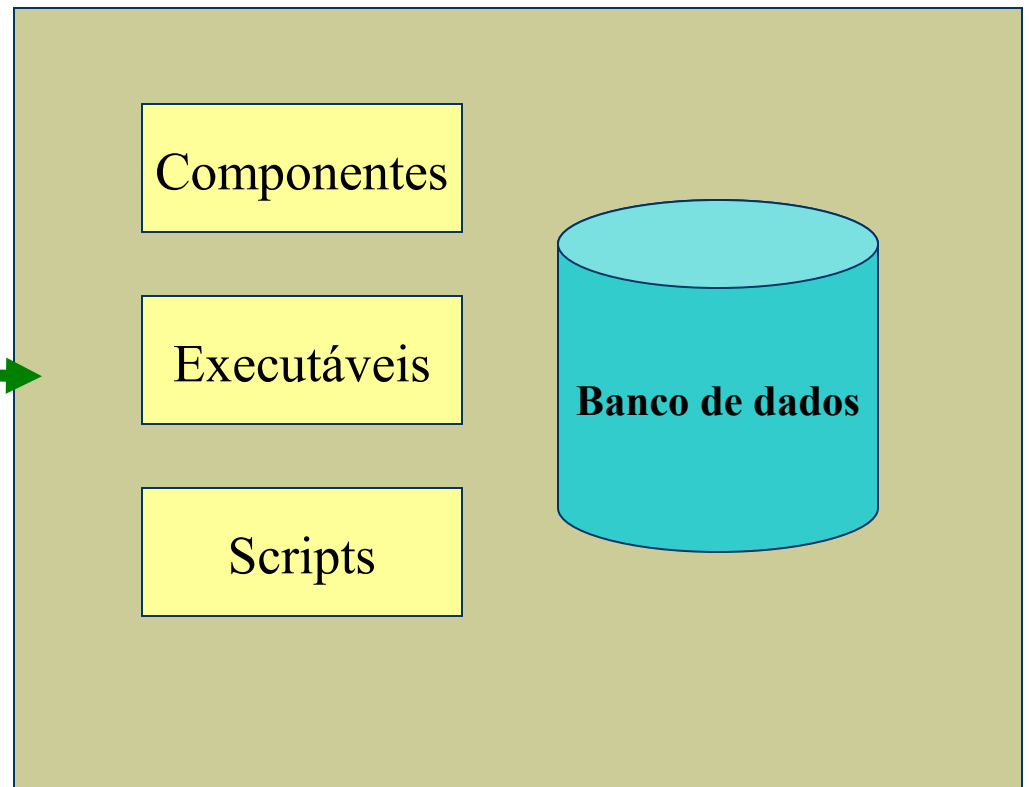


Elementos típicos da Internet thick client

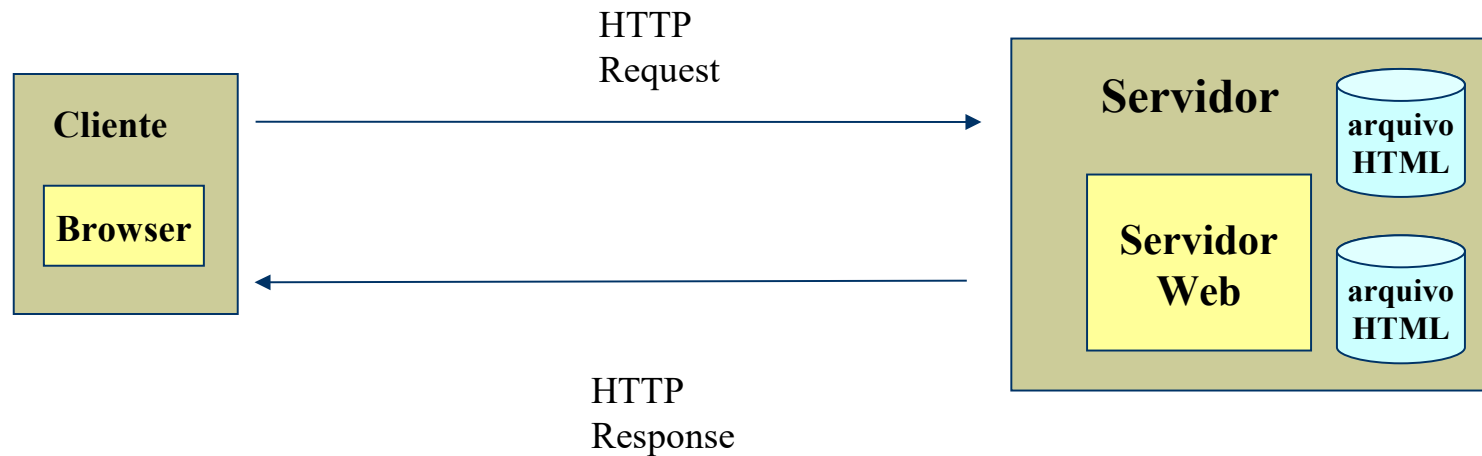
Cliente



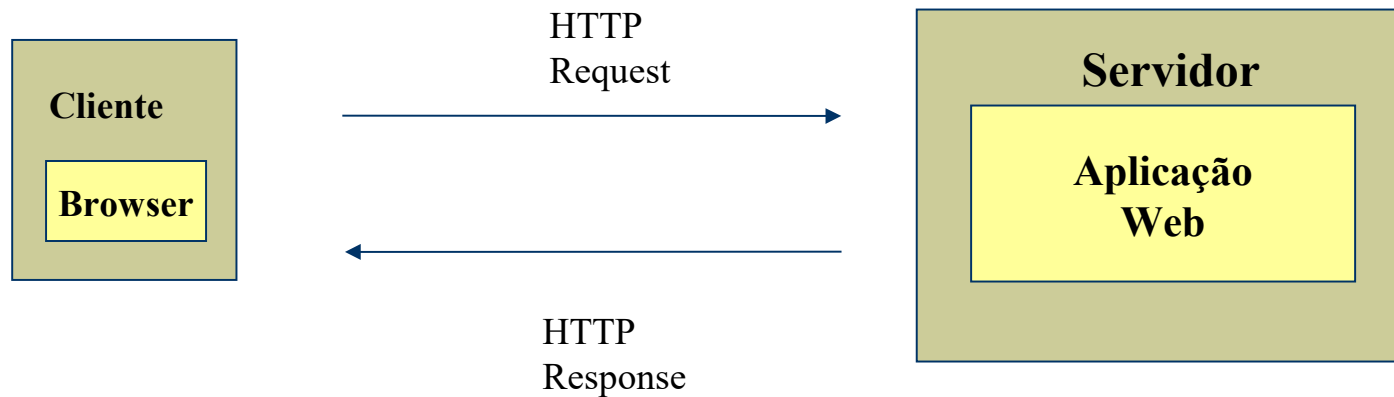
Servidor



Como as páginas *web* estáticas trabalham



Como as páginas *web* dinâmicas trabalham



O protocolo HTTP

- ◆ Protocolo de rede situado em uma camada acima da TCP/IP.
- ◆ Possui características específicas para aplicações baseadas na Web.
- ◆ A estrutura de um diálogo do tipo HTTP é uma simples seqüência de operações *request/response*.
- ◆ O *web browser* faz o *request* e o *web server* responde.

Elementos chaves de um *request*

- ◆ Ação a ser realizada. Representada por um dos métodos do HTTP.
- ◆ Página que desejamos obter acesso (URL).
- ◆ Parâmetros do formulário invocado.

Elementos chaves de um *response*

- ◆ Código de retorno do *request*.
- ◆ Tipo do conteúdo retornado(texto, figura, HTML etc).
- ◆ Conteúdo(o texto HTML, a figura etc).

Retornando um HTML

Cabeçalho HTTP

```
<html>
<head>
...
</head>
<body>
<img src=...>
</body>
</html>
```

Conteúdo
HTTP

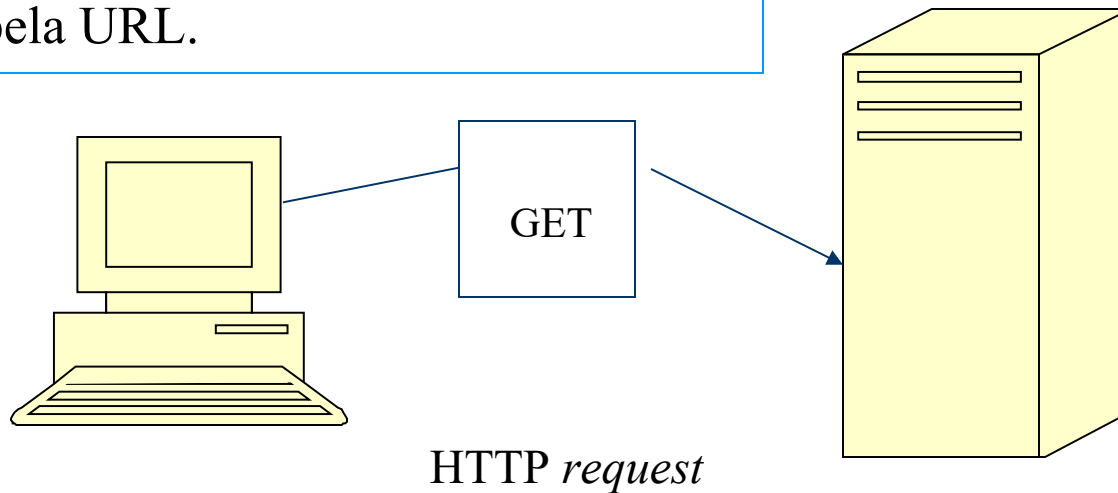
Gera um outro *request*.

Métodos utilizados pelo *request*

- ♦ Um *request* solicita serviços ao *web server* através de métodos do protocolo HTTP.
- ♦ Métodos do HTTP: **GET**, **POST**, HEAD, TRACE, PUT, DELETE, OPTIONS e CONNECT.
- ♦ O *Web browser* envia um HTTP **GET** para o servidor solicitando um recurso. Pode ser: uma página HTML, um JPEG, um PDF etc.
- ♦ O **POST** pode solicitar um recurso e, ao mesmo tempo, enviar um formulário com dados.
- ♦ O **GET** envia dados pela URL!

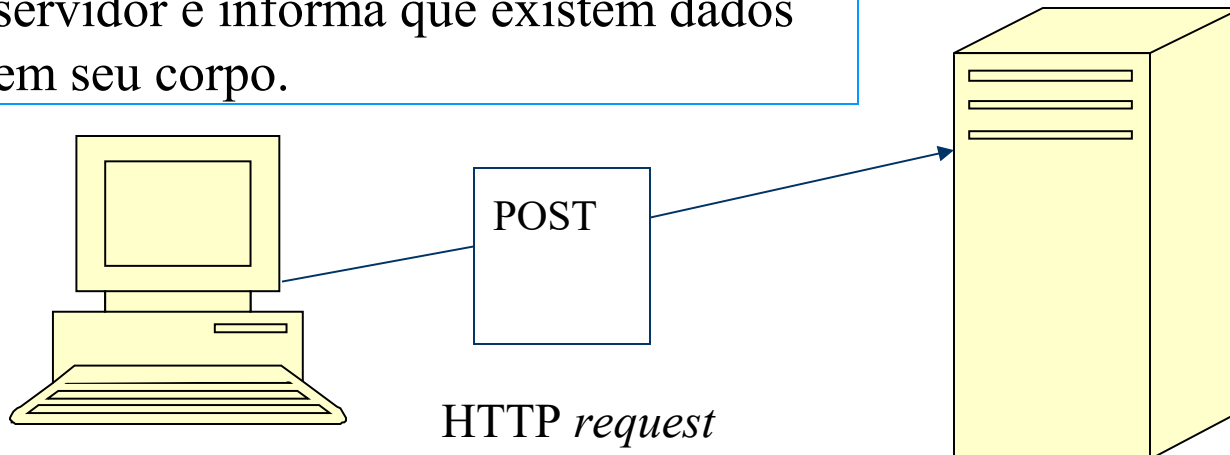
HTTP *request* GET

O GET pede uma página ao servidor passando os parâmetros Agência=3080 e Conta=123456 pela URL.



HTTP *request* POST

O POST pede uma página ao servidor e informa que existem dados em seu corpo.



HTTP *response*

- ◆ Composto de um cabeçalho e de um corpo.
- ◆ O cabeçalho serve para informar ao *web browser* :
 - Se a requisição foi ou não bem sucedida.
 - O tipo do conteúdo que está sendo passado(conhecido como MIME *type*).
- ◆ O corpo contém o conteúdo que será renderizado pelo *web browser*.

Fluxo primário de uma operação *request response*

- ◆ Usuário seleciona uma URL.
- ◆ *Web browser* cria um HTTP GET *request*.
- ◆ O HTTP GET é enviado para o *Web server*.
- ◆ O *Web server* localiza a página solicitada.
- ◆ *Web server* gera um HTTP *response*.
- ◆ O HTTP *response* é enviado para o *Web browser*.
- ◆ O *Web browser* renderiza o HTML.

Uniform Request Locator (URL)

◆ <http://www.nce.ufrj.br:80/concursos/login.html>

Protocolo

Nome do Servidor.
Possui um IP address.

Porta da Aplicação.
Default=80

Nome do recurso
solicitado.
Default=index.html

Caminho onde
o servidor vai
localizar o recurso.

Obs.: Caso seja utilizado o método GET
a URL conterá os parâmetros que serão
passados para o servidor.

Que código Java escrevemos para a *web*?

- ♦ *Servlets*.
- ♦ JavaServer Pages(JSP).
- ♦ Classes de negócio.
- ♦ Classes de acesso ao banco de dados.

Servlets

- ♦ Introduzidos pela Sun em 1996 com o propósito de acrescentar conteúdo dinâmico aos aplicativos *web*.
- ♦ Um *servlet* é uma classe Java executada por um *container*.
- ♦ Tem como benefícios: bom desempenho, portabilidade, rápido ciclo de desenvolvimento e robustez.

Servlet

```
import javax.servlet.*;  
import javax.servlet.http.*;
```

```
public class PrimeiroServlet extends HttpServlet {
```

```
    public void doGet(HttpServletRequest request,  
        HttpServletResponse response)  
        throws IOException, ServletException{
```

```
        PrintWriter out = response.getWriter();
```

```
        out.println("<HTML>");  
        out.println("<HEAD>");  
        out.println("<TITLE>Java para web com servlet e JSP</TITLE>");  
        out.println("</HEAD>");  
        out.println("<BODY>");  
        out.println("BemVindo ao curso de Java para web");  
        out.println("</BODY>");  
        out.println("</HTML>");
```

```
    }
```

```
}
```

JavaServer Page

◆ Exemplo 1: HTML puro

```
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
Java para web  
</BODY>  
</HTML>
```

◆ Exemplo 2: HTML + código Java = JSP

```
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
<%  
    out.println("Java para web");  
%>  
</BODY>  
</HTML>
```

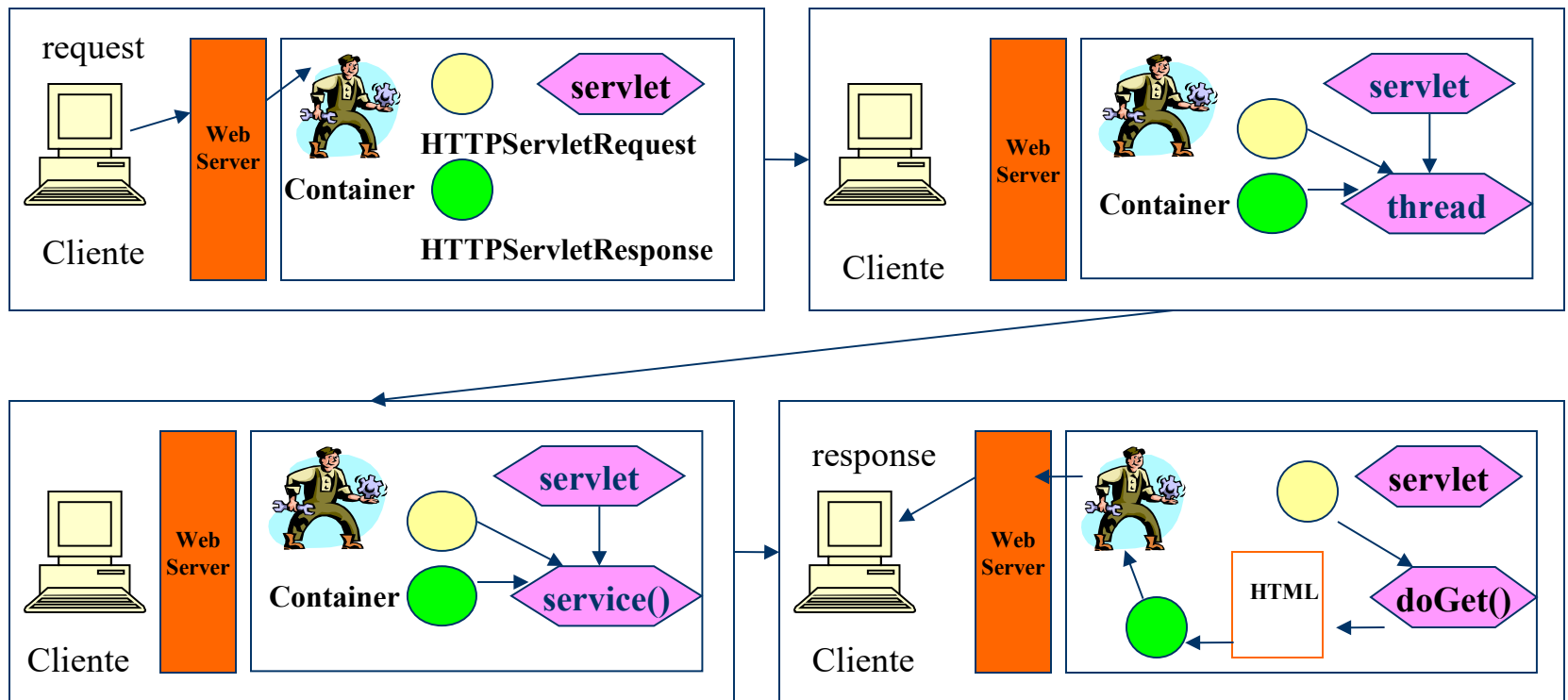
Container

- ♦ O *web server* não sabe tratar páginas dinâmicas.
- ♦ Necessidade de um *container* para abrigar *servlets* e JSPs.
- ♦ O TomCat é um dos mais populares *containers* do mercado.
- ♦ O *web server* solicita ao *container* as páginas dinâmicas.
- ♦ *Servlets* e JSPs não possuem um método `main()`. São carregados pelo *container*.

Container - Propósitos

- ◆ Suporte à comunicação de alto nível. Isenta os desenvolvedores de *servlets* de escreverem *sockets*.
- ◆ Administra o ciclo de vida dos *servlets*.
- ◆ Suporte à múltiplas *threads*.
- ◆ Suporte à segurança. Transparente para o desenvolvedor.
- ◆ “Transforma” um JSP em um *servlet*.
- ◆ Pode atuar também como *web server*.

Container – Tratando requests



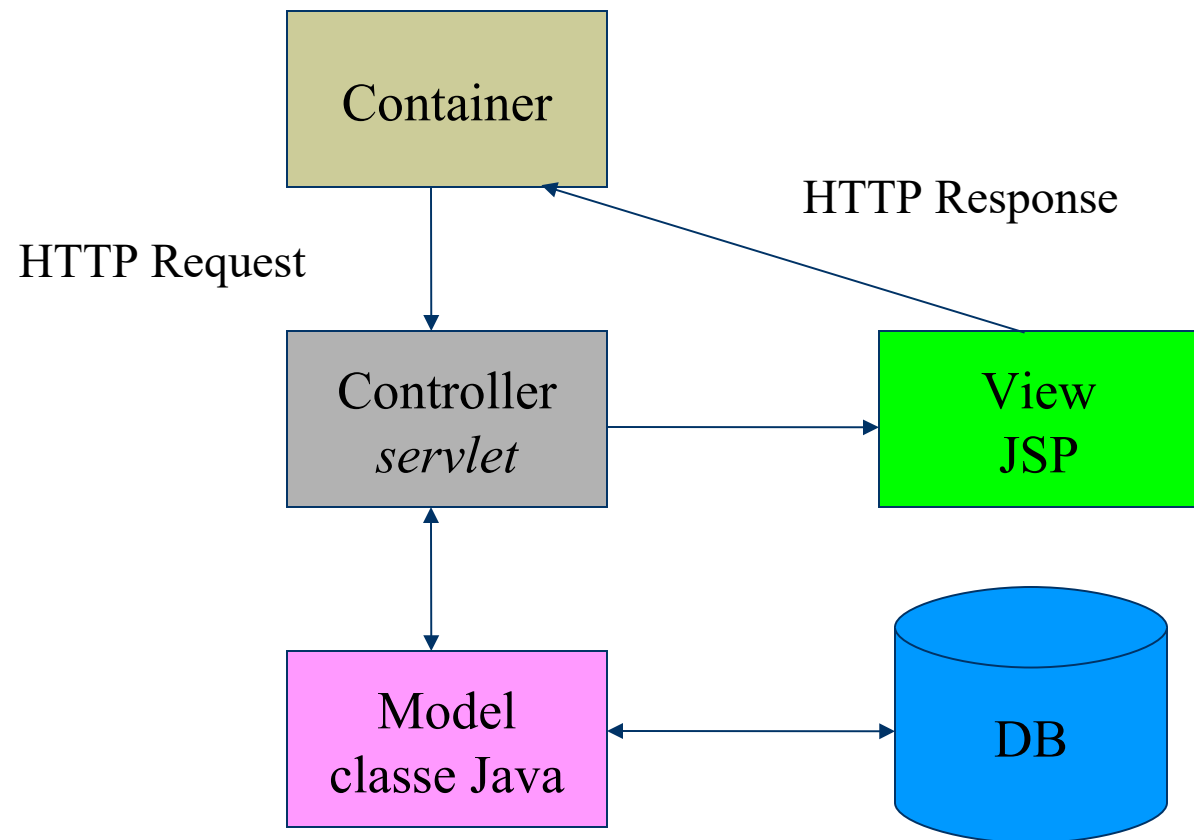
Uma aplicação *web* em Java



Model View Controller (MVC)

- ◆ Padrão de projeto empregado nas aplicações *web*.
- ◆ Separa a lógica do negócio da apresentação.
- ◆ A lógica fica em classes Java específicas.
- ◆ Possibilita o reuso destas classes por outros aplicativos.
- ◆ Divide mais claramente as responsabilidades:
 - A classe é o Model.
 - O *servlet* é o Controller.
 - A JSP é a View.

Model View Controller (MVC)



Model View Controller (MVC)

- ♦ O **Controller** recebe os dados do cliente, critica-os, e os repassa ao Model.
- ♦ O **Model** aplica as regras do negócio e retorna a informação para quem as solicitou.
- ♦ A **View** obtém o estado do Model, repassado pelo Controller, apresentando-o ao cliente.

Criando uma aplicação *web*

- ◆ São necessários 4 passos:
 1. Definir as páginas que serão vistas pelo cliente.
 2. Criar o ambiente de desenvolvimento.
 3. Criar o ambiente de produção/distribuição.
 4. Realizar os testes.

Aplicação *Sugestão Musical*

Visão do cliente

nonononononononononononononononononnn

Selecione o estilo musical preferido:

Rock: ☒

Samba: ☐

Ópera: ☐

MPB: ☐

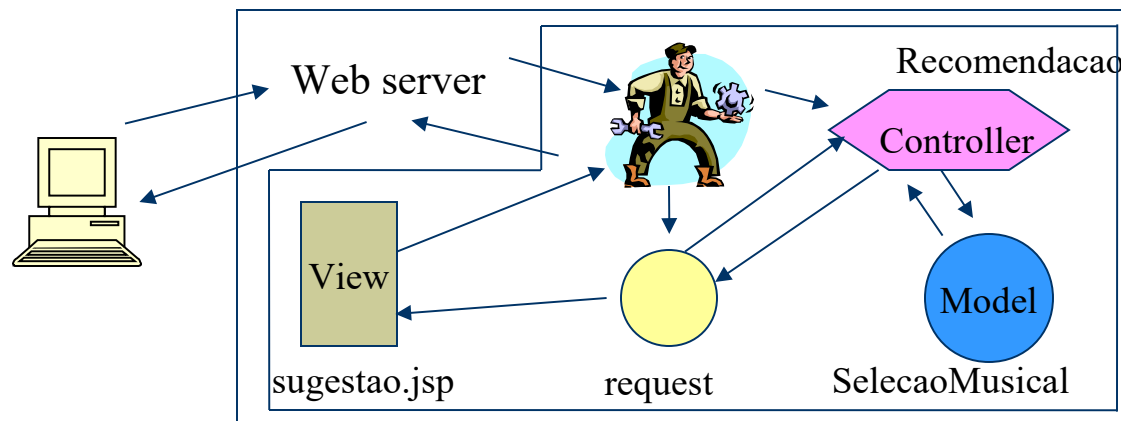
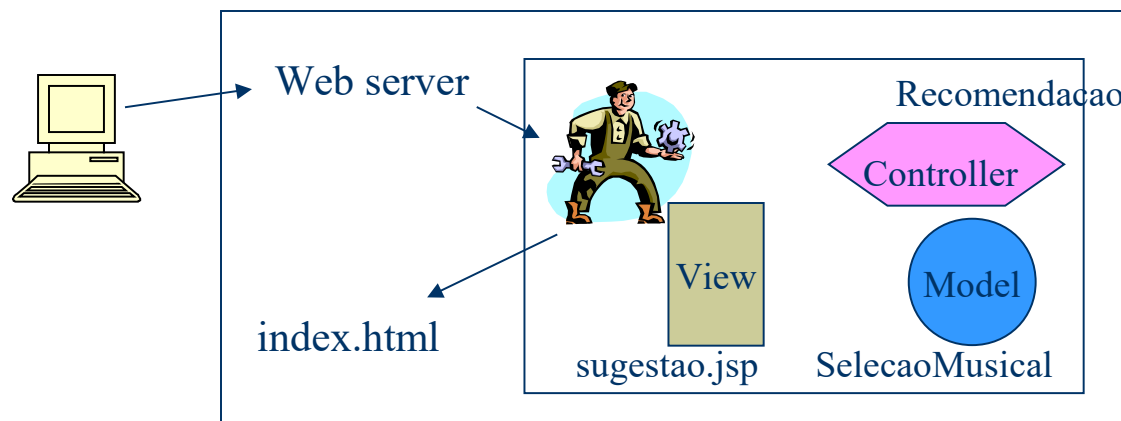
Enviar

nononononononononononononononononnn

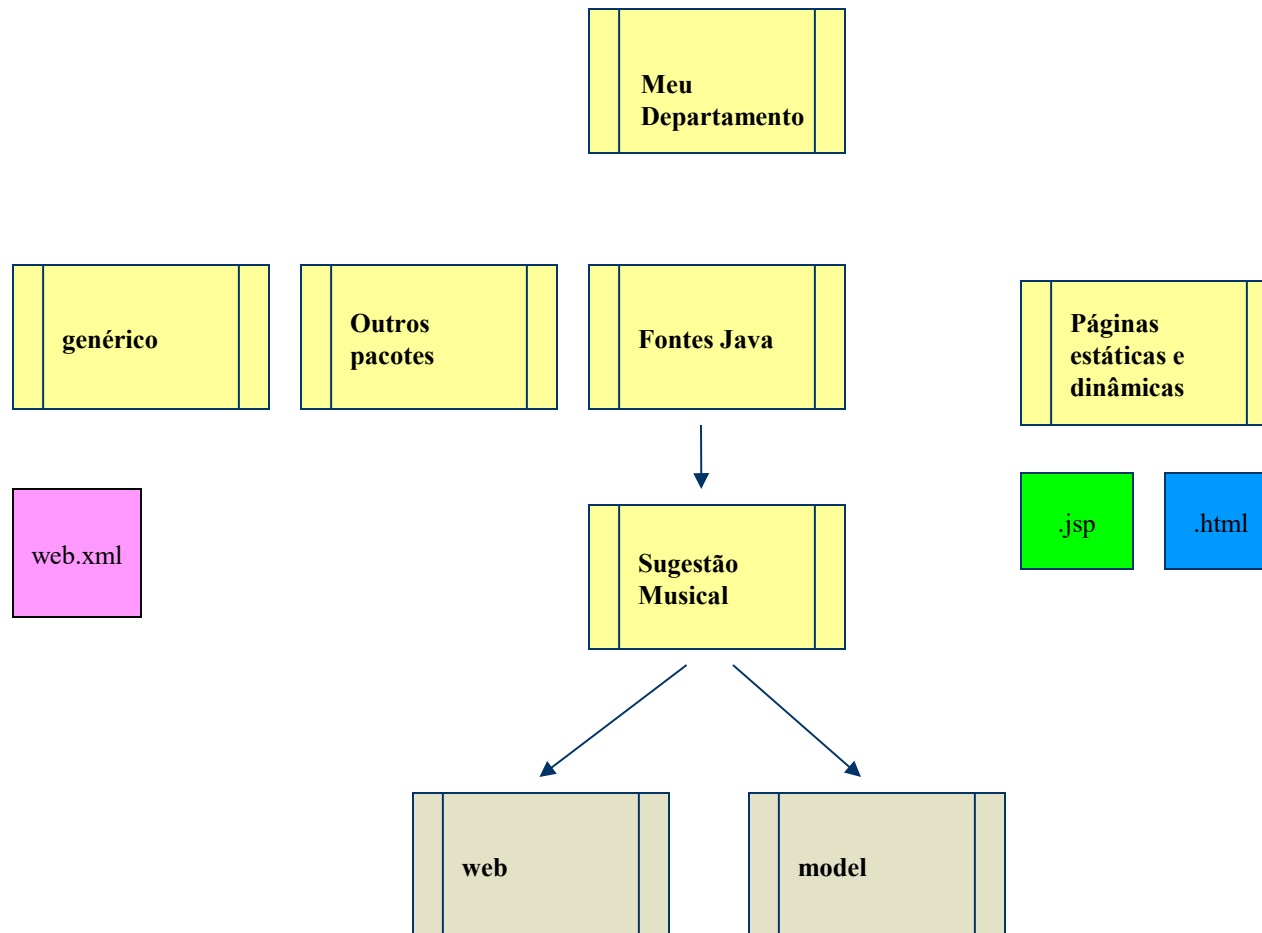
Recomendação musical :

Led Zeppelin
U2
The Who
Yes

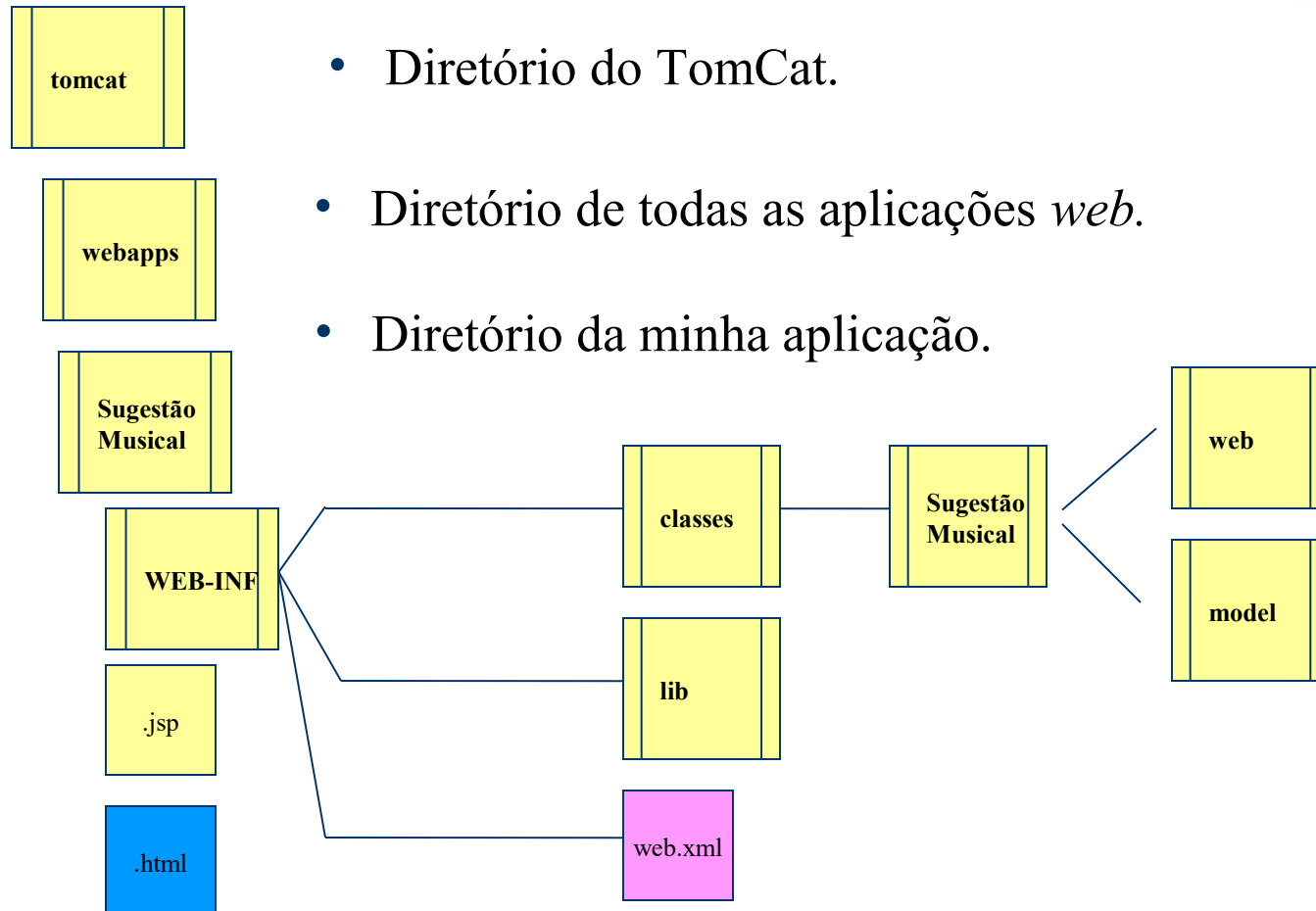
Arquitetura do aplicativo



Ambiente de desenvolvimento



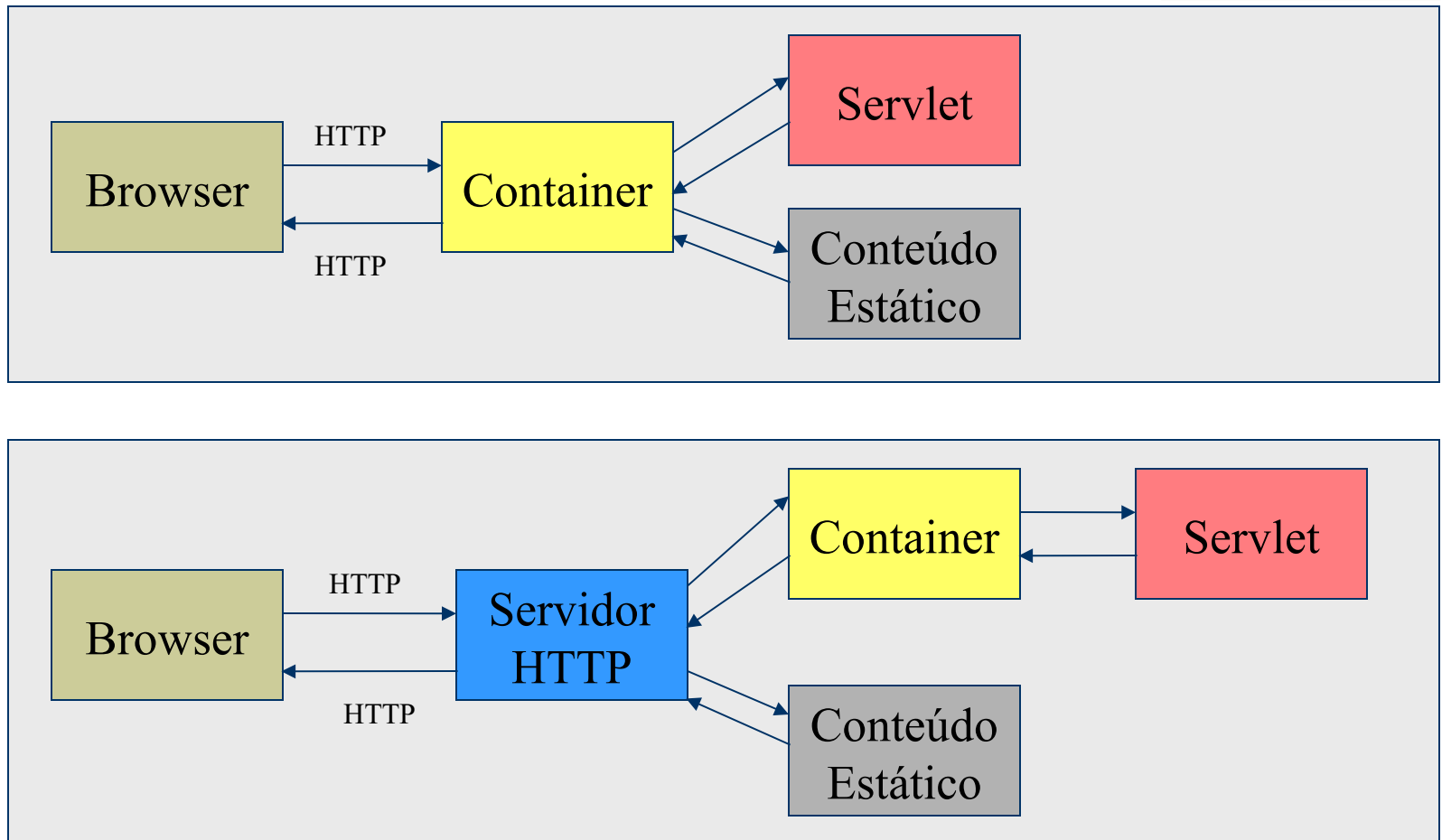
Ambiente de testes



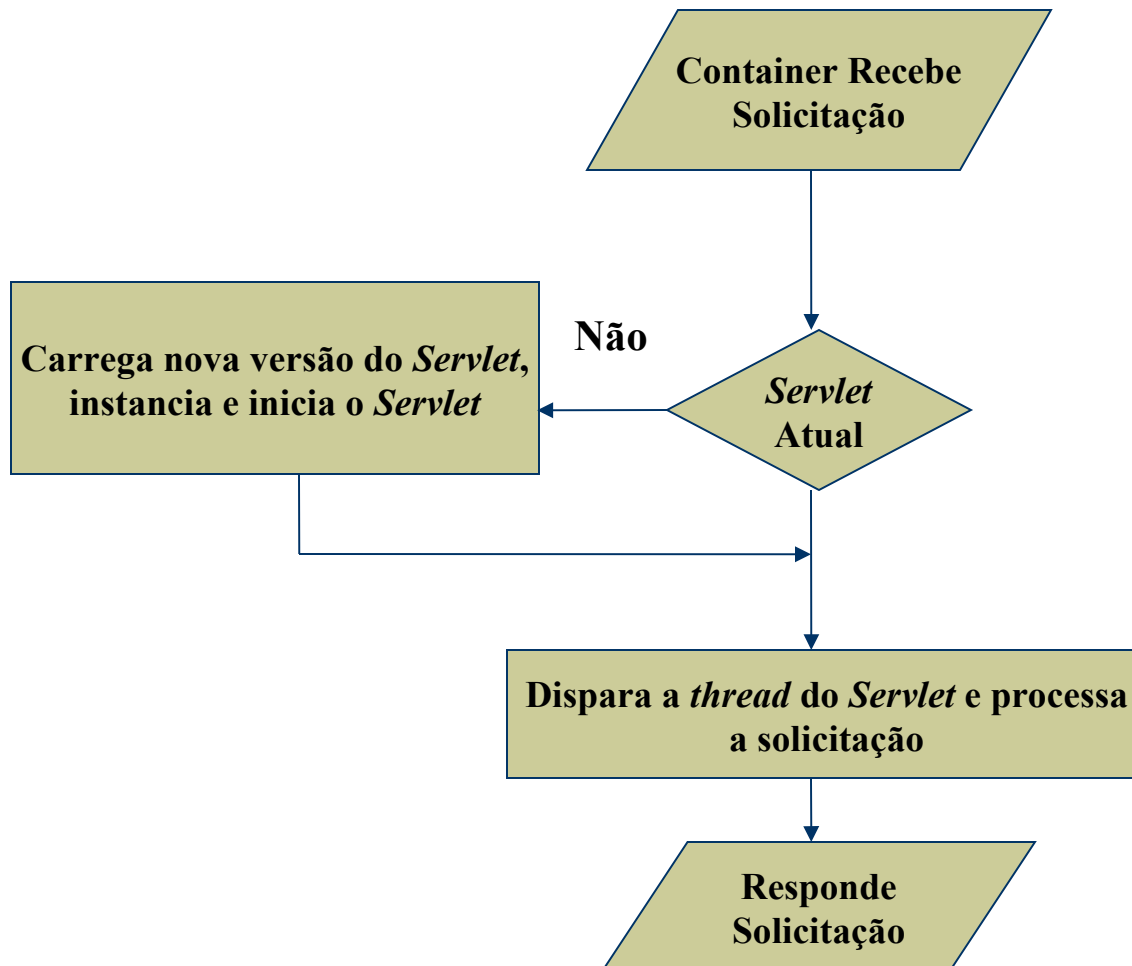
Características do sub-diretório WEB-INF

- ◆ Não fica visível para o *web browser* cliente.
- ◆ Residência do arquivo descritor `web.xml`.
- ◆ Os *servlets* residirão no sub-diretório `classes`.
- ◆ As classes que refletem as regras do negócio também residirão no sub-diretório `classes`.

Ativando um *servlet*



A carga de um *servlet*



A distribuição descritiva

- ◆ É um documento XML que contém informações que descrevem os servlets.
- ◆ Denominado web.xml.
- ◆ Possui a *tag* web-app que descreve todos os servlets da aplicação.
- ◆ Associados a cada servlet têm as *tags* <servlet-name> , <servlet-class> e <servlet-mapping>.
- ◆ <servlet-name> é o nome que o Tomcat irá referenciar o servlet.
- ◆ <servlet-class> é o nome efetivo do servlet sem a extensão *.class*.

A distribuição descritiva <servlet-mapping>

- ◆ Associa um URL a cada servlet.
- ◆ Evita que o nome do servlet seja apresentado no *web browser*.
- ◆ Utiliza a *tag* <url-pattern>.
- ◆ <url-pattern> define um nome que estará associado ao servlet desejado.

A distribuição descritiva

```
<web-app>
  <servlet>
    <servlet-name>Login</servlet-name>
    <servlet-class>LoginServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Login</servlet-name>
    <url-pattern>/loginservlet</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>MinhaCompra</servlet-name>
    <servlet-class>CompraServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MinhaCompra</servlet-name>
    <url-pattern>/minhacompra</url-pattern>
  </servlet-mapping>
</web-app>
```

Como invocar um *servlet* ?

- ◆ Invocando um servlet :
 - <http://localhost:8080/minhaapp/loginservlet>
 - <http://www.dcc.ufrj.br/minhaapp/minhacompra>
- ◆ Form *tags* para invocar um servlet:
 - `<form action=“../loginservlet” method=“get”>`
 - `<form action=“../minhacompra” method=“post”>`

A página *index.html*

```
<html><body>
<h1 align="center" >Selecione o estilo musical preferido:</h1>
<form method="POST" action="EscolhaGrupo">
<select name="estilo" size="1">
<option> Rock
<option>Samba
<option> Opera
<option> MPB
</select><br>
<center>
<input type="SUBMIT" value="Enviar" >
</center>
</form>
</body>
</html>
```

O servlet Recomendacao

```
package SugestaoMusical.web;
import SugestaoMusical.model.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class Recomendacao extends HttpServlet{
    public void doPost(HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException {

        String estilo = request.getParameter("estilo");
        SelecaoMusical selecao = new SelecaoMusical();
        ArrayList<String> retorno = selecao.getLista(estilo);
        request.setAttribute("listaRecomendada" , retorno);

        RequestDispatcher vista = request.getRequestDispatcher("sugestao.jsp");

        vista.forward(request, response);

    }
}
```

A classe *SelecaoMusical*

```
package SugestaoMusical.model;

import java.util.*;
public class SelecaoMusical{

    public ArrayList getLista(String estilo){

        ArrayList<String> grupos = new ArrayList<String>();
        if (estilo.equals("Rock")){
            grupos.add("Led Zeppelin");
            grupos.add("The Who");
            grupos.add("U2");
            grupos.add("Yes");
        }
        else if (estilo.equals("Samba")){
            grupos.add("Zeca Pagodinho");
            grupos.add("Fundo de Quintal");
            grupos.add("Dona Ivone Lara");
            grupos.add("Martinho da Vila");
        }
    }
}
```


A classe *SelecaoMusical*

```
else if (estilo.equals("Opera")){  
    grupos.add("Placido Domingo");  
    grupos.add("Luciano Pavarotti");  
    grupos.add("Jose Carreras");  
    grupos.add("Enrico Caruso");
```

```
}
```

```
else {
```

```
    grupos.add("Chico Buarque");  
    grupos.add("Milton Nascimento");  
    grupos.add("Ellis Regina");  
    grupos.add("Gonzaguinha");
```

```
}
```

```
return grupos;
```

```
}
```

```
}
```

A JSP *sugestão*

```
<%@ page import="java.util.*" %>
<html>
<body>
  <h1 align =center="center"> Recomendação Musical: JSP</h1>
  <% ArrayList<String> estilo =(ArrayList) request.getAttribute("listaRecomendada");
    for (String musica:estilo){
      out.print("<br>" + musica);
    }
  %>
</body>
</html>
```

O descritor *web.xml*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE web-app
```

```
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
```

```
  "http://java.sun.com/dtd/web-app_2_3.dtd">
```

```
<web-app>
```

```
  <servlet>
```

```
    <servlet-name>Musicas</servlet-name>
```

```
    <servlet-class>SugestaoMusical.web.Recomendacao</servlet-class>
```

```
  </servlet>
```

```
  <servlet-mapping>
```

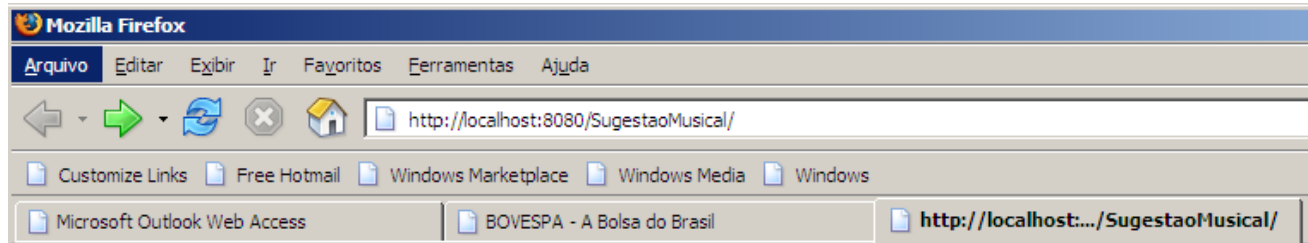
```
    <servlet-name>Musicas</servlet-name>
```

```
    <url-pattern>/EscolhaGrupo</url-pattern>
```

```
  </servlet-mapping>
```

```
</web-app>
```

O resultado



Selecione o estilo musical preferido:

Rock

Enviar



Recomendação Musical: JSP

Led Zeppelin
The Who
U2
Yes

Como codificar um formulário HTML

- ♦ Um formulário contém uma ou mais formas de entradas de dados tais como: *text boxes*, botões, *check boxes*, e *list boxes*.
- ♦ Um formulário **deve conter pelo menos um controle** tal como o botão SUBMIT.
- ♦ Qualquer dado associado ao controle será passado para o *servlet* ou para a JSP que está identificada pelo URL do atributo Action.

Como codificar um formulário HTML

- ♦ Tag `<form>` `</form>` define o início e o fim do formulário.
- ♦ Possui os seguintes atributos:
 - **Action** – especifica o URL do servlet ou da JSP que será chamada quando o usuário clicar o botão SUBMIT.
 - **Method** – especifica que método do protocolo HTTP será usado na operação de *request*. Pode ser GET ou POST.

Uso dos métodos GET e POST

- ◆ Quando usar o método GET ?
 - Se quiser transferir dados mais rapidamente.
 - Se o formulário HTML possui menos de 4 KB de tamanho.
 - Se não há problemas em os parâmetros aparecerem no URL.
- ◆ Quando usar o método POST ?
 - Se estiver transferindo mais do que 4 KB de tamanho.
 - Se não é conveniente os parâmetros aparecerem no URL.

Como codificar um formulário HTML

- ◆ Tag `<input>` define o tipo da entrada.
- ◆ Atributos comuns:
 - **Name** – é o nome do tipo.
 - **Value** – é o valor *default* do controle.

Como codificar um formulário HTML exemplo

- ◆ Código de um formulário HTML e seu resultado

```
<p>Um form que contém duas text boxes e um botão.</p>

<form action="confirma.jsp" method="post">
  <p>
    Nome:<input type="text" name="nome"><br>
    Email:<input type="text" name="sobrenome">
    <input type="submit" value="submit">
  </p>
</form>
```

Um form que contém duas text boxes e um botão.

Nome:

Email:

Como codificar *text boxes*, *passwords* e campos *hidden*

- ◆ Atributos dos controles de texto:
 - **Type** – especifica o tipo do controle de entrada para os *text boxes*.
 - **Name** – especifica o nome do controle. Este é o nome que será utilizado pela aplicação JSP ou servlet.
 - **Value** – especifica o valor do dado no controle.
 - **Size** – especifica o tamanho do campo de controle em caracteres.
 - **Maxlength** – especifica o número máximo de caracteres que pode estar contido no campo.

Tipos válidos para os *text boxes*

- ◆ Um tipo **Text** cria um *text box* padrão.
- ◆ Um tipo **Password** apresenta um *box* com asteriscos.
- ◆ Um tipo **Hidden** cria um campo *hidden* que armazena textos que não são apresentados pelo *browser*.

Exemplos de *text boxes*, *passwords* e campos *hidden*

```
<p>Login:  <input type="text" name="login" value="jsilva"></p>  
<p>Senha:  <input type="password" name="senha" value="112358"></p>  
<input type="hidden" name="codigoProduto" value="jr01"><br>
```

Login:

Senha:

Como codificar botões

- ◆ Atributos dos botões:
 - **Type** – especifica o tipo do controle de entrada. Os tipos aceitáveis são Submit, Reset ou Button.
 - **OnClick** – especifica o método JavaScript que será executado quando Button for clicado.

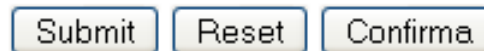
Tipos válidos para os botões

- ◆ O tipo **Submit** ativa o atributo Action do formulário.
- ◆ O tipo **Reset** inicia todos os controles do formulário com seus valores originais.
- ◆ O tipo **Button** cria um botão **JavaScript** que quando acionado executa um método pré-estabelecido.

Exemplos do uso de botões

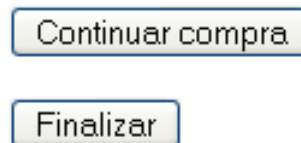
- 3 tipos de botões:

```
<input type="submit" value="Submit">  
<input type="reset" value="Reset">  
<input type="button" value="Confirma" onClick="validate(this.form) ">
```



- 2 botões Submit na mesma página:

```
<form action="compra/index.jsp" method="post">  
  <input type="submit" value="Continuar compra">  
</form>  
<form action="servlet/compra.ServletClient" method="post">  
  <input type="submit" value="Finalizar">  
</form>
```



Como codificar *checkboxes* e *radiobuttons*

- ◆ Atributos destes botões:
 - **Type** – especifica o tipo de controle. Os tipos aceitáveis são Checkbox ou Radio.
 - **Checked** – seleciona previamente determinado controle.

Exemplos de *radiobuttons* e *checkboxes*

```
<input type="checkbox" name="addEmail" checked>
```

Sim, me adicione na lista de emails.


```
<br>
```

Entrar em contato por:


```
<input type="radio" name="contatoPor" value="Email">Email
```

```
<input type="radio" name="contatoPor" value="Correios">Correios
```

```
<input type="radio" name="contatoPor" value="Ambos">Ambos<br>
```

```
<br>
```

Me interesse pelos seguintes estilos musicais:


```
<input type="checkbox" name="rock">Rock<br>
```

```
<input type="checkbox" name="classica">Samba<br>
```

```
<input type="checkbox" name="pagode">Pagode<br>
```

☒ Sim, me adicione na lista de emails.

Entrar em contato por:

☐ Email ☐ Correios ☐ Ambos

Me interesse pelos seguintes estilos musicais:

☐ Rock

☐ Samba

☐ Pagode

Como codificar *comboboxes* e *listboxes*

- ◆ Utiliza dois tipos de *tags*: **Select** e **Option**.
- ◆ Deve haver pelo menos uma *tag* Select e duas *tags* Option.
- ◆ Inicia com a *tag* Select que conterà as *tags* Option.
- ◆ A *tag* Option especifica as diferentes opções disponíveis no *box*.
- ◆ A *tag* Select possui o atributo Multiple que converte um *combox* em um *listbox*.
- ◆ A *tag* Option possui o atributo Selected que seleciona previamente uma opção.

Exemplos de *comboboxes* e *listboxes*

- Código de um *combobox*:

```
Selecione um país:<br>
<select name="pais">
  <option value="Brasil" selected>Brasil
  <option value="Canada">Canadá
  <option value="Mexico">México
</select>
```

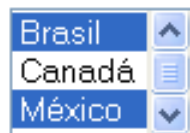
Selecione um país:



- Alterando para um *listbox*:

```
<select name="pais" multiple>
```

Selecione um país:



(Para selecionar mais de um país, pressione e segure a tecla Ctrl)

Como codificar uma *textarea*

- ◆ Uma *textarea* difere-se de uma *textbox* pelo fato de suportar múltiplas linhas.
- ◆ Usa a tag `<Textarea> </Textarea>`
- ◆ Atributos da *textarea*:
 - Rows – especifica o número de linhas visíveis na *textarea*. Se exceder é utilizado um *scroll bar*.
 - Cols – especifica a largura da *textarea*.

Exemplo de *textarea*

- Código de uma *textarea*:

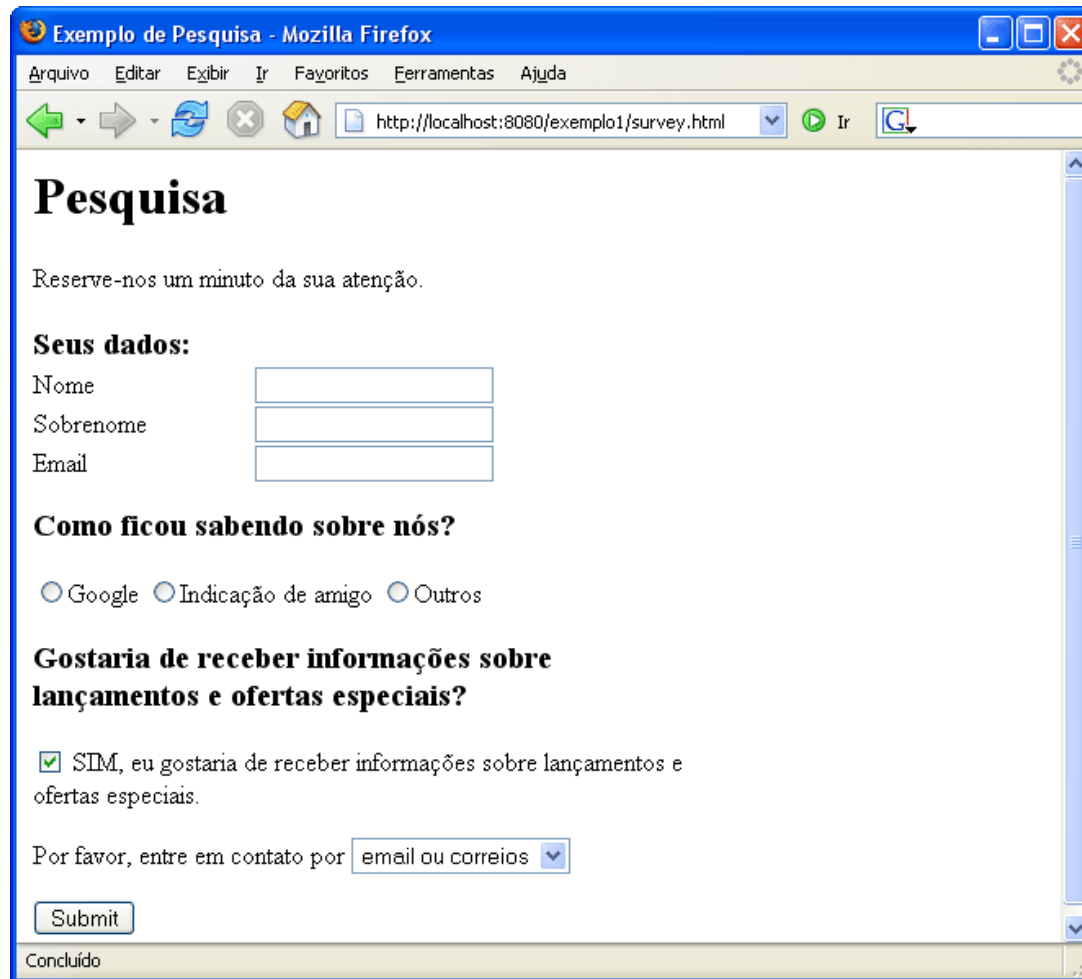
Comentários:


```
<textarea name="comentario" rows="8" cols="60"></textarea>
```

Comentários:

Sim, compatriotas, não esperemos mais, a hora é esta. Vamos cometer um haraquiri coletivo, (...). Pronto, aí tudo fica perfeito. Talvez um pouco esquisito, mas objeto inquestionável de admiração internacional e mais uma vez pioneiro: seremos o primeiro país sem povo e todos os problemas desapareceriam. Por que não pensamos nisso antes? Erram, como sempre, os catastrofistas. O Brasil tem futuro, sim, apesar de que não estaremos aqui para testemunhá-lo, mas não se pode querer tudo neste mundo." (João Ubaldo

Combinando *tags* - resultado final



The screenshot shows a Mozilla Firefox browser window with the title 'Exemplo de Pesquisa - Mozilla Firefox'. The address bar displays 'http://localhost:8080/exemplo1/survey.html'. The page content includes a title 'Pesquisa', a request for attention, a form for personal data (Nome, Sobrenome, Email), a question about how the user found the site (Google, Indicação de amigo, Outros), a question about receiving information (SIM, eu gostaria de receber informações sobre lançamentos e ofertas especiais.), and a contact preference dropdown (email ou correios). A 'Submit' button is at the bottom, and a status bar shows 'Concluído'.

Exemplo de Pesquisa - Mozilla Firefox

Arquivo Editar Exibir Ir Favoritos Ferramentas Ajuda

← → ↺ × 🏠 Ir

Pesquisa

Reserve-nos um minuto da sua atenção.

Seus dados:

Nome

Sobrenome

Email

Como ficou sabendo sobre nós?

☐ Google ☐ Indicação de amigo ☐ Outros

Gostaria de receber informações sobre lançamentos e ofertas especiais?

☒ SIM, eu gostaria de receber informações sobre lançamentos e ofertas especiais.

Por favor, entre em contato por

Concluído

Combinando *tags* – código HTML

```
<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
  <title>Exemplo de Pesquisa</title>
</head>
<body>

<table border="0" cellpadding="0">
<form action="/exemplo1/servlet/br.ufrj.dcc.poo.SurveyServlet" method="post">
  <tr>
    <td width="410" valign="top" colspan="2">

      <h1>Pesquisa</h1>
      <p>Reserve-nos um minuto da sua atenção.</p>
      <h3>Seus dados:</h3>
    </td>
  </tr>
  <tr>
    <td><p>Nome</p>

    <td><input type="text" name="nome" size="20" tabindex="1"></td>
  </tr>
  <tr>
    <td><p>Sobrenome</p>
    <td><input type="text" name="sobrenome" size="20" tabindex="2"></td>
  </tr>
```

Combinando *tags* – código HTML

```
<tr>
  <td><p>Email</td>

  <td><input type="text" name="email" size="20" tabindex="3"></td>
</tr>
<tr>
  <td colspan="2" height="12"></td>
</tr>
<tr>
  <td width="410" valign="top" colspan="2">
    <h3>Como ficou sabendo sobre nós?</h3>

    <p>
      <input type="radio" name="heardFrom" value="Google" tabindex="4">Google
      <input type="radio" name="heardFrom" value="Amigo">Indicação de amigo
      <input type="radio" name="heardFrom" value="Outros">Outros
    </p>
    <h3>Gostaria de receber informações sobre lançamentos e ofertas especiais?</h3>
    <p><input type="checkbox" name="querAtualiza" checked> SIM, eu gostaria de receber
informações sobre lançamentos e ofertas especiais.<br>
  </p>
  <p>
```


Combinando *tags* – código HTML

```
Por favor, entre em contato por
<select name="contatoPor">
  <option value="Ambos" checked>email e correios
  <option value="Email">email apenas
  <option value="Correios">correios apenas
</select>
</p>
<p><input type=submit value="Submit" tabindex="5"></p>
</td>
</tr>
</form>

</table>
</body>
</html>
```

O ciclo de vida de um *servlet*

◆ O método `init()`

- Inicia o *servlet*.
- O *container* chama este método apenas uma vez.
- Pode ser utilizado para iniciar variáveis, carregar o *driver* de um banco de dados etc.
- Recebe, através do objeto `ServletConfig`, os valores especificados no arquivo `web.xml`.
- Assinatura do método:

`public void init(ServletConfig config) throws ServletException`

- Método de uso opcional.

O ciclo de vida de um *servlet*

- ◆ O método `Service()`
 - É acionado pelo *container* após o término bem sucedido do método `init()`.
 - Executado a cada chamada do *servlet*.
- ◆ `Destroy()`
 - Remove o *servlet*. Ocorre por falta de uso ou *shutdown* do *server*.

Como desenvolver *servlets*

- ♦ Um *servlet* herda da classe `HttpServlet` que herda da classe `GenericServlet` que implementa a interface `Servlet`.
- ♦ Necessário importar os pacotes `javax.servlet`, `javax.servlet.http`.
- ♦ O método `init()` pode ser sobreposto.
- ♦ Pelo menos um método de serviço precisa ser sobreposto.

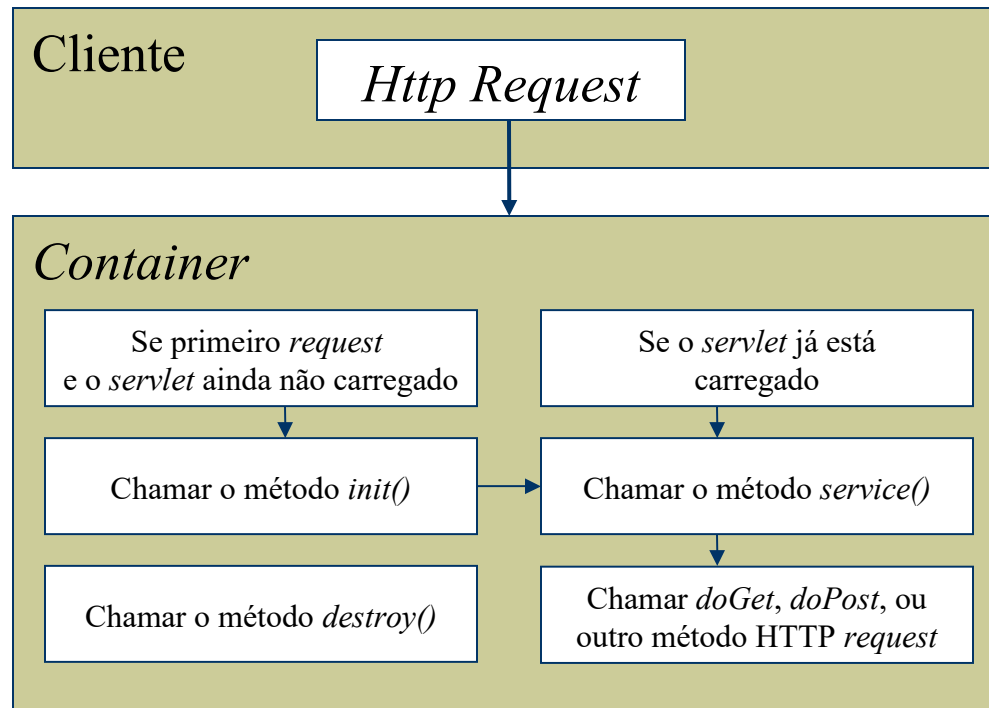
Como desenvolver *servlets*

- ◆ O método doGet processa todos os HTTP *requests* que usam o método Get.
- ◆ O método doPost processa todos os HTTP *requests* que usam o método Post.
- ◆ Estes métodos recebem os objetos *request* e *response* repassados pelo *container*.
- ◆ O método setContentType, do objeto *response*, indica o tipo de resposta retornada ao *browser*.
- ◆ O método getWriter, do objeto *response*, é usado para enviar o arquivo HTML para o *web browser*.

Alguns privilégios dos *servlets*

- ◆ Capacidade de “*logar*” eventos.
- ◆ Obter referências para outros recursos.
- ◆ Passar atributos para outros *servlets*.

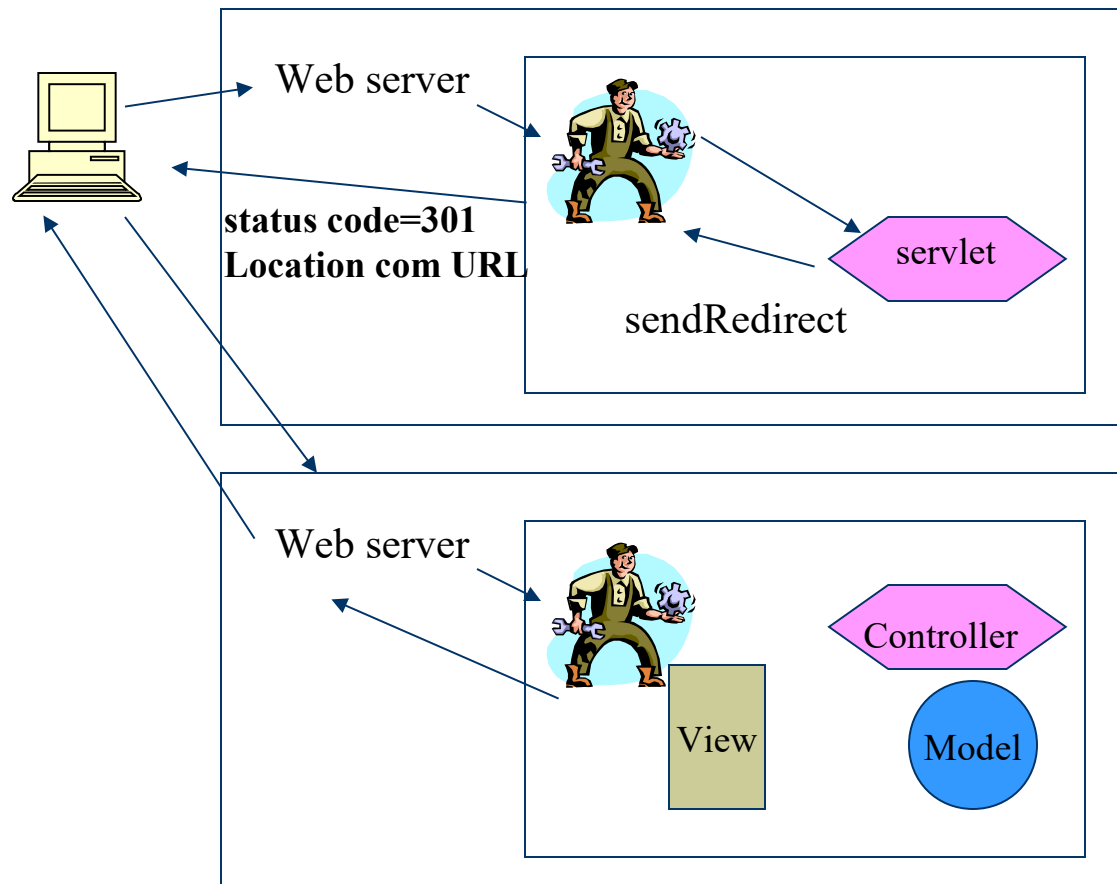
Como o *container* trata um *request* para um *servlet* ?



Redirecionando as respostas

- ◆ O *servlet* pode direcionar uma requisição para outro *servlet* ou para uma JSP.
- ◆ O *servlet* ou JSP destino pode residir em uma URL remota ou no mesmo *container*.
- ◆ O recurso remoto não tem acesso aos objetos *request* e *response* do *servlet* original.

Redirecionando para outra URL



Redirecionando para outra URL

◆ O que escrevo para redirecionar?

- `response.sendRedirect("http://www.nce.ufrj.br");`

ou

- `response.sendRedirect("/OutraAplicacao/Sugestao")`
 - Desvia para `http://www.nce.ufrj/ OutraAplicacao/Sugestao`;
 - ◆ A "/" significa conectar-se à raiz (outra webapps).

ou

- `response.sendRedirect("FechaCompra/Sugestao");`
 - Conecta-se à webapps original.

- ◆ Obs.: A URL do novo destino é apresentada no *web browser*.

Redirecionando para o mesmo local

- ♦ O que escrevo para redirecionar?
 - `RequestDispatcher vista = request.getRequestDispatcher("sugestao.jsp");`
 - `vista.forward(request,response);`
- ♦ O *web browser* desconhece este redirecionamento.

ServletConfig

- ◆ Objeto criado pelo *container* e utilizado para passar parâmetros de iniciação para um *servlet*.
- ◆ Parâmetros são definidos no web-xml.
- ◆ Evita a inserção de valores, passíveis de alterações, nos *servlets*.
- ◆ Para ativar uma nova versão web-xml é só fazer um *redeploy* no *container*.
- ◆ Existe apenas um por cada *servlet*.
- ◆ Não pode ser alterado.

ServletConfig

- ◆ Oferece alguns dos seguintes métodos:
 - **getInitParameter(String)**
 - Retorna o conteúdo de um parâmetro específico.
 - **Enumeration getInitParameterNames()**
 - Retorna um conjunto com os nomes dos parâmetros especificados.

ServletConfig

◆ Especificando no web-xml:

```
<servlet>
  <servlet-name>Musicas</servlet-name>
  <servlet-class>com.exemplo.web.Recomendacao</servlet-class>
  <init-param>
    <param-name>faleConosco</param-name>
    <param-value>centralatendimento@nce.ufrj.br</param-value>
  </init-param>
  <init-param>
    <param-name>areaVendas</param-name>
    <param-value>vendasatendimento@nce.ufrj.br</param-value>
  </init-param>
</servlet>
```

• Obtendo no *servlet* ou JSP:

```
getServletConfig().getInitParameter("faleConosco");
getServletConfig().getInitParameter("areaVendas");
```

ServletContext

- ◆ Reflete o ambiente onde o *servlet* é executado.
- ◆ Criado pelo *container* para cada aplicativo *web* existente.
- ◆ Utilizado para os *servlets* compartilharem informações.
- ◆ Independe de sessão.
- ◆ Suporta atributos que podem ser modificados ou recuperados pelos *servlets* ou JSPs.

ServletContext

- ◆ Permite a declaração de parâmetros no web-xml.
- ◆ Estes parâmetros podem ser recuperados, em qualquer instante, pelos *servlets* ou JSPs.
- ◆ Lembrete: Atributos retornam um *Object* e parâmetros retornam um *String*.

ServletContext

- ◆ Oferece alguns dos seguintes métodos:
 - **getAttributeNames()**
 - Retorna um conjunto com os nomes dos atributos armazenados.
 - **getAttribute(String)**
 - Retorna um atributo específico do contexto.
 - **setAttribute(String, Object)**
 - Armazena um atributo no contexto
 - **removeAttribute(String)**
 - Remove um atributo do contexto.

ServletContext

- **getInitParameter(String)**
 - Retorna o conteúdo de um parâmetro específico.
- **Enumeration getInitParameterNames()**
 - Retorna um conjunto com os nomes dos parâmetros especificados.
- **getRequestDispatcher(String)**
 - Desvia para um recurso local.

ServletContext

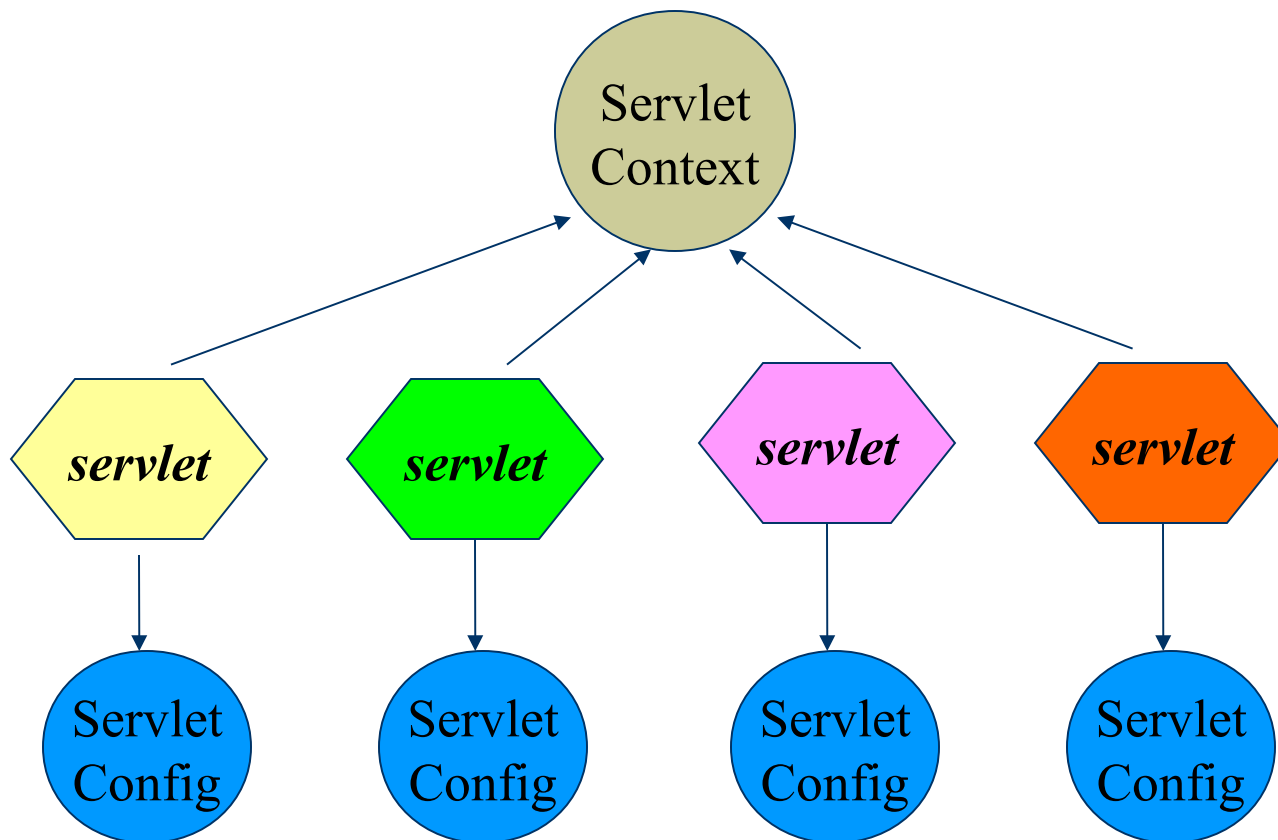
◆ Especificando no web-xml:

```
<servlet>
  <servlet-name>Musicas</servlet-name>
  <servlet-class>com.exemplo.web.Recomendacao</servlet-class>
  <init-param>
    <param-name>faleConosco</param-name>
    <param-value>centralatendimento@nce.ufrj.br</param-value>
  </init-param>
  <init-param>
    <param-name>areaVendas</param-name>
    <param-value>vendasatendimento@nce.ufrj.br</param-value>
  </init-param>
</servlet>
<context-param>
  <param-name>enderecoReal</param-name>
  <param-value>Avenida Rio Branco 156</param-value>
</context-param>
```

ServletContext

- Obtendo um parâmetro pelo *servlet* ou JSP:
`getServletContext().getInitParameter("endereçoReal");`
- Criando um atributo pelo *servlet* ou JSP:
`getServletContext().setAttribute("endereco", "Avenida Rio Branco 156");`
- Obtendo o atributo pelo *servlet* ou JSP:
`getServletContext().getAttribute("endereco");`

ServletConfig e ServletContext



O ServletContextListener

- ◆ Permite que a aplicação seja notificada quando um objeto **ServletContext** é **criado** ou **destruído**.
- ◆ Controla os eventos do ciclo de vida do contexto.
- ◆ Aplicação pode recuperar os parâmetros de iniciação do ServletContext.
- ◆ Acesso ao ServletContext é feito através do objeto **ServletContextEvent**.
- ◆ Inserido no diretório acima do *web* e do *model*.

O ServletContextListener

- ◆ Possui dois métodos:
 - **contextInitialized(ServletContextEvent evento)**
 - Acionado quando o contexto é criado.
 - **contextDestroyed(ServletContextEvent evento)**
 - Acionado quando o contexto é destruído.

O ServletContextListener

- Especificando no web-xml:

```
<web-app>
  <servlet>
    <servlet-name>Musicas</servlet-name>
    <servlet-class>com.exemplo.web.Recomendacao</servlet-class>
    <init-param>
      <param-name>faleConosco</param-name>
      <param-value>centralatendimento@nce.ufrj.br</param-value>
    </init-param>
  </servlet>
  <context-param>
    <param-name>enderecoReal</param-name>
    <param-value>Avenida Rio Branco 156</param-value>
  </context-param>
  <listener>
    <listener-class>
      com.exemplo.IniciaMinhaAplicacao
    </listener-class>
  </listener>
</web-app>
```


O ServletContextListener

```
import javax.servlet.ServletContext;
import javax.servlet.ServletContextListener;
import javax.servlet.ServletContextEvent;

public class AppLifecycleEvent implements ServletContextListener {

    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("Initializing Application !");
        // Load the JDBC driver
        try {
            Class.forName("org.gjt.mm.mysql.Driver ");
        }
        catch (ClassNotFoundException e) {
            System.out.println(e.toString());
        }

        // Get the ServletContext object
        ServletContext servletContext = sce.getServletContext();

        // Set a ServletContext attribute
        servletContext.setAttribute("dbUrl", "jdbc:mysql://Fred");
        System.out.println("Application initialized");
    }

    public void contextDestroyed(ServletContextEvent cse) {
        System.out.println("Application shut down");
    }
}
```

Código extraído do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

Servlet obtendo acesso ao contexto

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class ApplicationEventDemoServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD>");
        out.println("<TITLE>Application Event Demo Servlet</TITLE>");
        out.println("</HEAD>");
        out.println("<BODY>");
        out.println("Your database connection is ");

        // get the ServletContext object
        ServletContext servletContext = getServletContext();
        // display the "dbUrl" attribute
        out.println(servletContext.getAttribute("dbUrl"));

        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

Código extraído do livro: Java para Web com Servlets, JSP e EJB, de Budi Kurniawan.

Fluxo de execução

- ◆ *Container* é iniciado e lê o web-xml.
- ◆ *Container* cria o ServletContext da aplicação.
- ◆ *Container* cria pares nome-valor dos parâmetros de iniciação do Contexto(se houver).
- ◆ *Container* dá ao ServletContext referências para estes pares.
- ◆ *Container* cria uma instância do *listener* da aplicação *web*.
- ◆ Método `contextInitialized()` é acionado e obtém a referência para o ServletContext.
- ◆ Código do método `contextInitialized()` cria os atributos de interesse.

Outros *listeners*

- ◆ ServletContextAttributeListener
- ◆ ServletRequestListener
- ◆ ServletRequestAttributeListener
- ◆ HttpSessionListener
- ◆ HttpSessionBindingListener
- ◆ HttpSessionAttributeListener
- ◆ HttpSessionActivationListener

Exercício

- ♦ Criar uma aplicação *web* que valide um *login* e uma senha.
- ♦ O *login* deverá ser: cursoJavaWeb
- ♦ E a senha : 123456
- ♦ Se entrada válida: apresenta o *login* e a senha para o usuário
- ♦ Se não OK: diz qual foi o erro.