

# Programação Interativa para Internet JSP/Java

Alcione de Paiva Oliveira  
Universidade Federal de Viçosa

1

## Sumário

---

- **Introdução.**
- **Arquiteturas de programas para a Web.**
- **Linguagens para a Web na camada cliente**
- **Linguagens para a Web na camada servidora**
- **Servlets**
- **JSP**

2

## Introdução

---

- A popularização da Internet criou um amplo campo para novas oportunidades de negócios e relações.
- Para responder a essas novas necessidades uma grande quantidade de programas tem sido desenvolvida tendo como meio a Internet.
- É um campo ainda novo e propício ao surgimento de novas idéias de programas que abrem, por sua vez, novos campos (ex: napster, icq, sites de busca, etc.)

3

## Introdução

---

### Tipos de aplicações

- Business-to-consumer (B2C)- Compras online de CDs, Livros, etc.
- Business-to-business (B2B)- Negócios entre empresas e fornecedores, etc.
- User-to-data - acesso à bases de informação.
- User-to-user - Chat, e troca de informações entre usuários (napster).

4

# Introdução

## Tipos de aplicações

- B2B (business to business) - entre empresas.  
Exemplo: uma empresa de automóveis comunica aos fornecedores de auto-peças a necessidade de se repor o estoque.



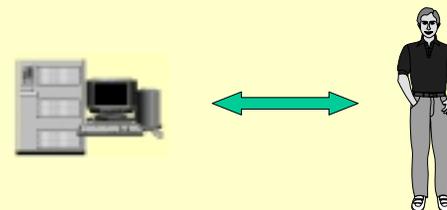
Obs: Neste tipo de aplicação a linguagem XML possui um papel muito importante.

5

# Introdução

## Tipos de aplicações

- B2C (business to customer) - entre empresa e consumidor.  
Exemplo: uma pessoa compra um livro na Internet.



6

# Introdução

- O desenvolvimento de programas na Internet, e na Web em particular, não passa por uma simples adaptação das técnicas de desenvolvimento de programas Cliente/Servidor.
- Existem diferentes requisitos que devem ser levados consideração no caso de projeto de programas para a Web:

7

# Introdução

## Comparação entre os requisitos de programas Cliente/servidor e programas para a Web.

	<i>Cliente/Servidor</i>	<i>Web</i>
<b>Arquitetura - No Camadas</b>	Em geral 2 camadas. Cliente e camada de acesso ao SGBD	3 ou mais camadas: Cliente, Servidor Web, servidor de Aplicação e camada de acesso ao SGBD
<b>Arquitetura - Cliente</b>	Cliente "gordo".	Cliente Magro
<b>Segurança</b>	Melhor controle sobre o acesso. Pode usar protocolos proprietários e contas do SGBD	Devido o uso de protocolos abertos (TCP/IP), HTTP e a visibilidade potencial de toda a Internet, deve-se usar firewalls, criptografia e sistemas de autenticidade. Em geral as contas de SGBD não podem ser usadas.
<b>Desempenho</b>	Número de acessos controlado. Pode prescindir de um pool de conexões.	Devido ao potencialmente grande número de acessos deve-se adotar balanceamento de carga e pool de conexões.

# Introdução

Além disso, um programa para a Web pode envolver diferentes linguagens na diferentes linguagens nas diferentes camadas e módulos:

- Camada cliente: XML, HTML, Javascript, DHTML, Java
- Camada Servidora: XML, Java, PHP, ASP, C/C++
- Camada do BD: linguagem para procedimentos armazenados: ex PL/SQL

9

# Introdução

No projeto de solução para a Web é importante avaliar os seguintes itens:

- Elementos da aplicação: cliente, servidores e serviços externos (ERP, sistema financeiros, etc).
- Tecnologia de suporte: JSP/Servlets, XML, PHP, ASP, DHTML, EJB, etc.
- Estrutura da aplicação: Número de camadas, padrão MVC
- Desempenho: Pool de conexões, balanceamento de carga, etc.
- Segurança: Firewalls, mecanismos de autenticação, criptografia

10

# Introdução

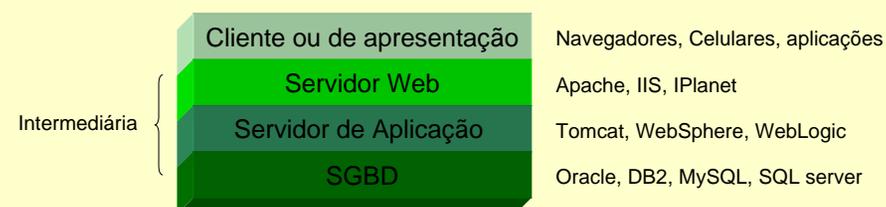
## Requisitos para Soluções para a Web

- Expandir a base de consumidores.
- Possuir regime de operação 24x7 (24 horas por dia, 7 dias por semana).
- Reduzir o custo da transação de vendas.
- Ser escalável.
- Possuir interfaces com os sistemas da empresa.
- Empregar novos métodos de pagamento na Internet (cartão de crédito, dinheiro eletrônico, etc.)
- Oferecer um suporte melhor para a pré e pós venda.

11

# Arquiteturas

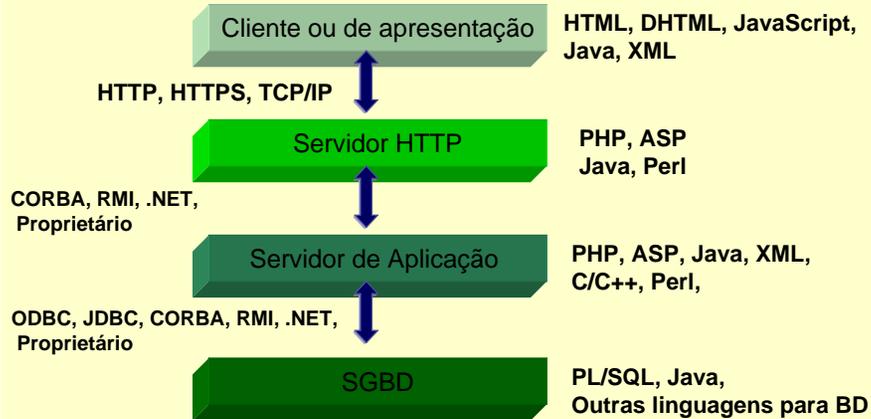
## Camadas



12

# Arquiteturas

## Camadas: Linguagens & Protocolos



13

# Arquiteturas

## Camada de Apresentação

- Pode ser dividida em cliente magro ou leve (thin) e cliente pesado.
- O cliente pesado pode ser caracterizado por programas em Java ou Delphi onde toda a lógica da apresentação (e parte da lógica do negócio) está codificada no programa. Podem comunicar diretamente com o servidor de aplicação ou SGBD.
- O cliente leve é caracterizado, tipicamente, por navegadores que recebem a codificação da interação em tempo de execução. Parte da lógica da apresentação pode ser processada no cliente (ex: javascript).

14

# Arquiteturas

## Camada de Apresentação

clientes pesados

### Vantagens

- Atualmente podem produzir interfaces mais elaboradas que os clientes magros.

### Desvantagens

- Maior dificuldade de atualização e manutenção. Necessidade de ir até o cliente para a instalação ou transmissão longa via rede.

15

# Arquiteturas

## Camada de Apresentação

clientes Leves

### Vantagens

- Maior controle sobre a segurança, distribuição e manutenção.
- Transmissão rápida via rede.
- Toda a lógica do negócio e parte da lógica da apresentação reside no servidor.

16

# Arquiteturas

## Camada Intermediária

### Servidor HTTP

- Responsável pelo recebimento das requisições HTTP (ex: POST, GET, etc).
- Pode atender a requisição isoladamente (no caso de páginas estáticas) ou encaminhar a requisição para um módulo ou servidor de aplicação (no caso de páginas dinâmicas).
- Deve ser capaz de ter um bom desempenho (atendimento de várias requisições simultâneas), ser robusto e seguro.
- Exemplos: Apache, IIS, IPlanet, IBM HTTP server.

17

# Arquiteturas

## Camada Intermediária

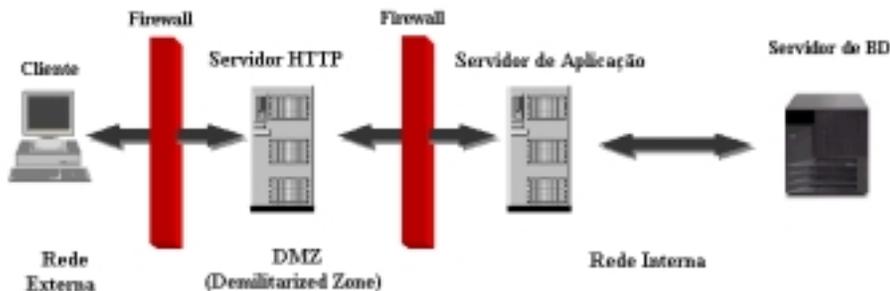
### Servidor de Aplicação

- Responsável pelo recebimento das requisições HTTP (ex: POST, GET, etc) e pela execução da aplicação responsável pelo atendimento da requisição.
- Para cumprir seus objetivos o SA fornece serviços de construção dinâmica de páginas, processamento da lógica de negócios, acesso a dados, integração com aplicações externas, gerência de sessões, balanceamento de carga, recuperação a falhas.
- As funções do SA e do Servidor HTTP se confundem, sendo que em algumas arquiteturas os dois grupos de funções são realizadas pelo mesmo software.
- Exemplos: Jakarta Tomcat, WebSphere, Web Logic, Orion

18

# Arquiteturas

## Topologia



19

# Arquiteturas

## Topologia

- DMZ - é onde os servidores HTTP são instalados. A DMZ é protegida da rede pública por um firewall, também chamado de *firewall de protocolo*. O firewall de protocolo deve ser configurado para permitir tráfego apenas através da porta 80.
- Um segundo firewall, também chamado de *firewall de domínio* separa a DMZ da rede interna. O *firewall de domínio* deve ser configurado para permitir comunicação apenas por meio das portas do servidor de aplicação.

20

# Arquiteturas

## Topologia 2



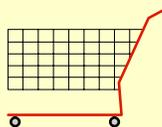
21

# Protocolos para CE

- O protocolo básico da Web é o HTTP, que faz parte da suite de protocolos TCP/IP.
- Este protocolo possui uma deficiência básica para a implantação do CE: a falta de controle de sessão.
- Ou seja, cada requisição é tratada isoladamente, sem ligação com requisições prévias.
- No CE é necessário acompanhar as ações do cliente (exemplo: metáfora do carrinho de compras).
- Para implantar o acompanhamento da *sessão* do usuário são usadas recursos como o uso de *cookies* e campos escondidos.

22

# Metáfora Básica para o B2C



- A metáfora mais utilizada no B2C é a do carrinho de compras.
- Nesta metáfora o cliente navega pelo site e seleciona os itens que deseja adquirir, colocando em um carrinho de compras virtual.
- Note que existe a necessidade de acompanhar a interação do usuário com o site (sessão).

23

# Linguagens no lado Cliente

- HTML (*HyperText Markup Language*)
- DHTML (*Dynamic HyperText Markup Language*)
- JavaScript
- XML (*eXtensible Markup Language*)
- Java (Applet)
- Active X

24

# Linguagens no lado Cliente

## HTML (*HyperText Markup Language*)

- É a linguagem de apresentação da Internet.
- Atualmente está na versão 4.
- A maioria dos navegadores são compatíveis com a versão 3.2.
- Vantagens
  - Amplamente utilizada
  - Fácil domínio
- Desvantagens
  - Muito genérica
  - Impossibilidade de customização

25

# Linguagens no lado Cliente

## DHTML (*Dynamic HyperText Markup Language*)

### HTML4+JavaScript = DHTML

- DHTML permite um alto grau de flexibilidade para a construção de interfaces.
- Em particular, DHTML inclui cascading style sheets (CSS) que permite o uso de diferentes fontes, margens e espaçamento para várias partes da página. Permite também o posicionamento em coordenadas absolutas dos elementos a serem exibidos.
- DHTML aumenta o nível de funcionalidade do HTML por meio de um modelo de objetos do documento e um modelo de eventos.
- O modelo de objetos permite que linguagens de script, como JavaScript, controlem partes da página. Por exemplo, textos e imagens podem ser movidas sobre a tela ou escondidas sob o comando de um script.

26

# Linguagens no lado Cliente

## DHTML (*Dynamic HyperText Markup Language*)

- A linguagem script também pode ser usada para mudar a cor de um objeto quando o cursor do mouse passa sobre ele.
- *Desvantagem*
  - Duas implementações diferentes: Netscape e Microsoft

27

# Linguagens no lado Cliente

## DHTML

### Exemplo de CSS

```
/* links */
a:link { text-decoration: none; color:blue; }
a:visited { text-decoration: none; color: blue; }
a:active { text-decoration: underline; color: red; }
a:hover { text-decoration: underline; color: blue; }

/* tables */
table.main { color: black; font-family: Verdana, Arial, Helvetica,
             sans-serif; font-size: 10pt; text-align: justify; }
td.title { background: #7FB2FF; font-family: Verdana, Arial, Helvetica,
           sans-serif; color: #FFFFFF; font-size: 14pt; text-align: center; }

/* End of stylesheet.css */
```

28

# Linguagens no lado Cliente

## JavaScript

- JavaScript, apesar do nome, não tem relação com a linguagem Java.
- A linguagem JavaScript é semelhante a C/C++ e foi desenvolvida pela Netscape, incorporada ao Navigator e, posteriormente, ao IE.
- JavaScript é uma linguagem interpretada, ou seja, o código é incluído dentro do próprio arquivo HTML, interpretado pelo browser e depois executado.

29

# Linguagens no lado Cliente

## JavaScript

- O objetivo de JavaScript é estender a capacidade das páginas HTML, permitindo a execução de código na camada cliente.
- Com JavaScript é possível criticar a entrada do usuário, criar animações e desenvolver interações mais sofisticadas.
- JavaScript possui limitações que garantem alguma segurança, como não permitir acesso a arquivos no disco da estação e impedir abrir conexões de rede com outros sistemas.
- Apesar destas limitações, JavaScript pode ser utilizada para ataques do tipo engenharia social e Denial of Service (DoS).

30

# Linguagens no lado Cliente

## JavaScript (exemplo)

```
<html>
<head>
  <script language="JavaScript">
    function pushbutton() { alert("Ola Mundo!"); }
  </script>
</head>
<body>
  <form>
    <input type="button" name="Button1" value="Aperte-me"
      onclick="pushbutton()">
  </form>
</body>
</html>
```

31

# Linguagens no lado Cliente

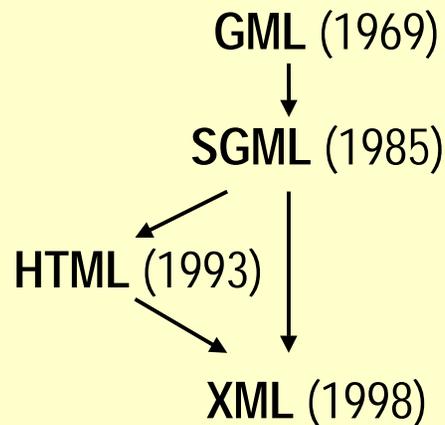
## XML (*eXtensible Markup Language*)

- XML permite que você especifique sua própria linguagem de marcação, sendo os tags definidos por um DTD (Document Type Definition).
- O conteúdo é produzido com o uso desses marcadores.
- Um conteúdo pode ser transformado em outro por meio do uso de especificações XSL (*eXtensible Stylesheet Language*).
- Atualmente apenas o IE5 é capaz de interpretar XML.

32

# Linguagens no lado Cliente

## Árvore da família



Datas da primeira publicação

33

# Linguagens no lado Cliente

## Componentes da tecnologia XML

XML é composta por várias linguagens e tecnologias de apoio, dentre elas:

XML- A linguagem básica, definindo um framework de no nível de metadados.

DTD - Estabelece um conjunto de restrições para um documento XML. Define a forma como o documento XML deve ser construído.

XSL (*eXtensible Stylesheet Language*) - Transforma um formato XML em outro. É composto por dois outros padrões: XSLT (*eXtensible Stylesheet Language Transformation*) e FO (*formatting objects*).

XSLT - define uma linguagem comum para transformar um documento XML em outro.

FO - define como exibir o resultado da transformação (semelhante ao CSS).

34

# Linguagens no lado Cliente

## Java (Applet)

- Java possibilita a construção de interfaces elaboradas.
- Apesar de ser uma solução robusta existem problemas no uso de Applets: o programador precisa contar com o fato do usuário possuir um navegador com suporte a Java e na versão apropriada.
- Você não pode contar com isso na Internet, principalmente se você deseja estender a todos o acesso às suas páginas. Em se tratando de Servlets, no lado do cliente pode existir apenas páginas HTML, evitando restrições de acesso às páginas.

35

# Linguagens no lado Cliente

## Exemplo Applet

```
import java.awt.*;
import java.applet.*;

public class Ola extends Applet
{
    public void init()
    {
        setSize(150,87);
        add(new Label("Ola Mundo!"));
    }
}
```

36

# Linguagens no lado Cliente

## Exemplo Applet

```
<html>
<applet
    code=ola.class
    width=400
    height=300>
</applet>
</html>
```

37

# Linguagens no lado Cliente

## Active X

- Componentes ActiveX são programas executáveis, desenvolvidos em uma linguagem de programação qualquer, como C, C++ ou Visual Basic, e encapsulados em um arquivo OCX.
- Um componente ActiveX, depois de carregado pelo browser, tem acesso irrestrito a qualquer recurso do seu sistema operacional e, até mesmo, ao hardware. Um controle ActiveX hostil pode infectar sua estação com um vírus, eliminar seus arquivos e desligar seu computador.

fonte: Luiz Paulo Maia email: lpmaia@training.com.br

38

# Linguagens no lado Cliente

## Active X

- O grande problema de segurança do ActiveX é seu mecanismo de autenticação, baseado no Autenticode. Qualquer desenvolvedor pode criar um componente ActiveX e solicitar uma assinatura digital para seu código. A assinatura é uma garantia dada por uma autoridade de certificação (Verisign por exemplo), que o código foi escrito realmente por seu autor e que não foi alterado.
- A assinatura garante a autenticidade e a integridade do código, mas não garante segurança. Um código pode ser autêntico e íntegro e mesmo assim conter um vírus ou desligar sua máquina

39

# Linguagens no lado Cliente

## Active X

- Quando o ActiveX é carregado, o browser verifica a assinatura do código e abre uma janela informando o nome de seu autor.
- Depois disto existem duas opções: aceitar o programa ou recusá-lo.
- A segurança do ActiveX está baseada inteiramente na decisão correta do usuário em aceitar ou não código. Se a escolha for errada, é possível que sua estação e toda a rede privada fiquem comprometidas.

40

# Linguagens no lado Servidor

- CGI (*Common Gateway Interface*)
- XML
- ASP (*Active Server Pages*). tecnologia desenvolvida pela Microsoft.
- PHP (*Personal Home Page*)
- Servlets/JSP (*Java Server Paggers*)
- ColdFusion

41

# Linguagens no lado Servidor

## XML

- XML e XSL podem ser usadas no lado servidor para codificar conteúdos e transformá-los para diferentes usuários. Permitindo o desenvolvimento de aplicações tanto para navegadores de PC como para dispositivos consumo eletrônico.
- O conteúdo é codificado em XML e analisador XML é usado para transformá-lo em uma saída baseado em um documento XSL.

42

# Linguagens no lado Servidor

## XML

- XML é também usado como meio para especificar o conteúdo de mensagens entre dois servidores, dentro da mesma empresa ou em um relacionamento B2B.
- O fator crítico aqui é acordo entre as partes sobre o esquema de codificação especificado em um documento DTD.
- Um analisador XML é usado para extrair conteúdo específico na mensagem.
- 
- O projeto deve considerar se será usada uma abordagem voltada para eventos, onde a API SAX é mais apropriada, se a estrutura em árvore do documento será navegada, por meio da API DOM.

43

# Linguagens no lado Servidor

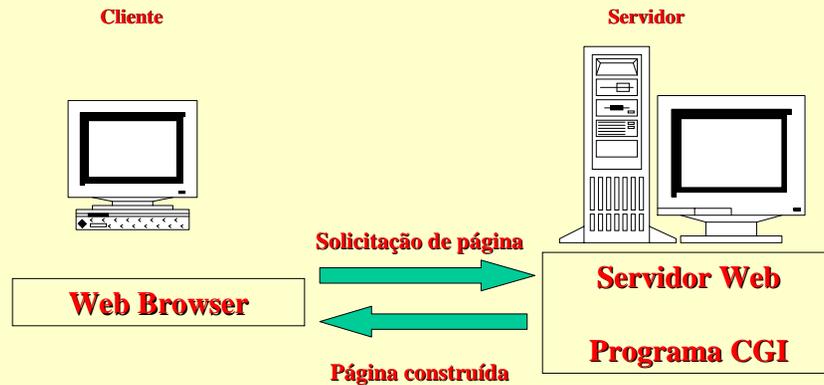
## CGI (*Common Gateway Interface*)

- Scripts CGI acionam programas no servidor.
- O uso de CGI sobrecarrega o servidor uma vez cada requisição de serviço acarreta a execução de um programa executável no servidor.
- Se houver algum erro na entrada de dados o CGI tem que produzir uma página HTML explicando o problema.
- Possuem baixa portabilidade e escalabilidade. Como ponto positivo suportam várias linguagens.

44

# Linguagens no lado Servidor

## CGI (Common Gateway Interface)



# Linguagens no lado Servidor

## PHP (Personal Home Page)

- É uma linguagem de scripts para HTML para ser executada no lado servidor.
- Criada em 1994 como um projeto pessoal de
- Atualmente está na versão 4
- Suporte extensivo à Banco de dados
  - ODBC, MySql, Sybase, Oracle...
- Syntax derived from C, Perl, Java, etc...
- Classes (OOP) and Functions

46

# Linguagens no lado Servidor

## PHP

- Incorporado ao Apache
  - Como um DSO – dynamic shared object
  - mod\_php
- Todos os outros servidores
  - Como binário CGI

47

# Linguagens no lado Servidor

## PHP (Exemplo)

```
<html>
  <body>
    <?php
      print "Ola mundo!";
    ?>
  </body>
</html>
```

48

# Linguagens no lado Servidor

## ASP(Active Server Pages)

- ASP é uma tecnologia da Microsoft® para criação de páginas dinâmicas na Web.
- Assim como PHP e JSP, trata-se da inclusão de scripts em meio a código HTML.
- Executa no IIS (Internet Information Server)
- Usa mais de uma linguagem (JavaScript or VBScript)

49

# Linguagens no lado Servidor

## ASP(Active Server Pages)

```
<%@ Language=VBScript %>
<html>
  <head>
    <title>Exemplo ASP</title>
  </head>
  <body>
    <%
      x = "Ola Mundo!"
    %>
    <%=x%>
  </body>
</html>
```

50

# Linguagens no lado Servidor

## Servlets

- Servlets são programas Java que executam em associação com servidores Web, atendendo requisições por informação na forma de páginas HTML.
- Servlets não possuem interface gráfica e podem trabalhar com vários tipos de servidores e não só com servidores Web, uma vez que a API dos Servlets não assume nada a respeito do ambiente do servidor.
- Os Servlets são carregados apenas uma vez e como são executados de forma multi-thread podem atender mais de uma mesma solicitação simultaneamente.

51

# Linguagens no lado Servidor

## Servlets (exemplo)

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

public class Ola extends HttpServlet {
  public String getServletInfo() { return "Ola versão 0.1"; }

  public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws IOException, ServletException {
    res.setContentType("text/html");
    PrintWriter os=res.getWriter();
    os.println("<html>");
    os.println("<head><title>Ola!</title></head>");
    os.println("<h1 align=center>Ola!</h1>");
    os.println("</body></html>");
    os.close();
  }
}
```

52

# Linguagens no lado Servidor

## JSP(Java Server Pages)

- Tecnologia desenvolvida pela Sun baseada em Servlets.
- JSP são páginas HTML que incluem código Java e outros *tags* especiais.
- Desta forma as partes estáticas da página não precisam ser geradas por `println`. Elas são fixadas na própria página. A parte dinâmica é gerada pelo código JSP.

53

# Linguagens no lado Servidor

## JSP(Exemplo)

```
<html>
  <head>
    <title>Exemplo JSP</title>
  </head>
  <body>
    <%
      String x = "Ola Mundo!"
    %>
    <%=x%>
  </body>
</html>
```

54

# Servlets

## A API Servlet

A API Servlet é composta por um conjunto de interfaces e Classes. O componente mais básico da API é interface Servlet. Ela define o comportamento básico de um Servlet.

```
public interface Servlet {
    public void init(ServletConfig config)
        throws ServletException;
    public ServletConfig getServletConfig();
    public void service(ServletRequest req,
        ServletResponse res)
        throws ServletException, IOException;
    public String getServletInfo();
    public void destroy();
}
```

55

# Servlets

## A API Servlet

O método `service()` é responsável pelo tratamento de todas das requisições dos clientes.

Os métodos `init()` e `destroy()` são chamados quando o Servlet é carregado e descarregado, respectivamente.

O método `getServletConfig()` retorna um objeto `ServletConfig` que contém os parâmetros de inicialização do Servlet.

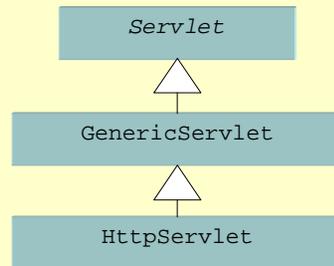
O método `getServletInfo()` retorna um `String` contendo informações sobre o Servlet, como versão e autor.

56

# Servlets

## A API Servlet

Tendo como base a interface Servlet o restante da API Servlet se organiza hierarquicamente como mostra a figura abaixo.



57

# Servlets

## A API Servlet

A classe `GenericServlet` implementa um servidor genérico e geralmente não é usada.

A classe `HttpServlet` é mais utilizada e foi especialmente projetada para lidar com o protocolo HTTP.

```
public abstract class HttpServlet  
extends GenericServlet  
implements java.io.Serializable
```

58

# Servlets

## A API Servlet

A classe derivada da `HttpServlet` deve sobrescrever pelo menos um dos métodos abaixo:

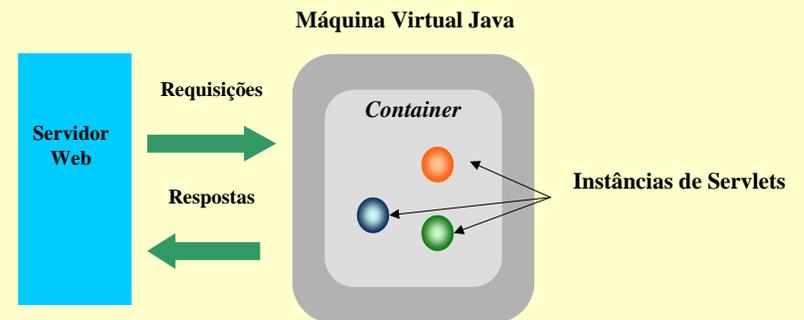
<code>doGet</code>	para tratar requisições HTTP GET.
<code>doPost</code>	para tratar requisições HTTP POST.
<code>doPut</code>	para tratar requisições HTTP PUT.
<code>doDelete</code>	para tratar requisições HTTP DELETE.

O método `service()`, que recebe todas as requisições, em geral não é sobrescrito, sendo sua tarefa direcionar a requisição para o método adequado.

59

# Servlets

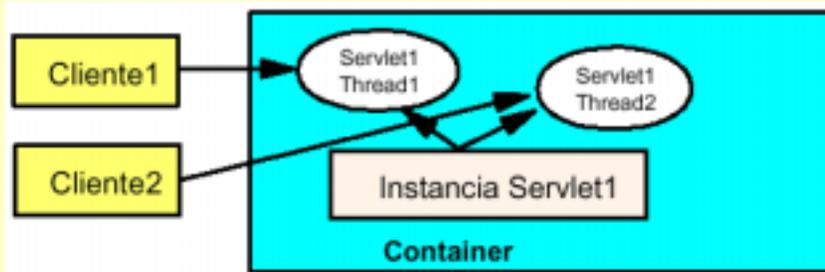
## Relacionamento entre Componentes



60

# Servlets

## Relacionamento entre Instâncias e Threads



61

# Servlets

## Exemplo

Para entendermos o que é um Servlet nada melhor que um exemplo simples. O exemplo a seguir gera uma página HTML em resposta a uma requisição.

62

# Servlets

## Exemplo

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

public class Ola extends HttpServlet
{
    public String getServletInfo() { return "Ola versão 0.1";}

    public void doGet(HttpServletRequest req,
        HttpServletResponse res)
        throws IOException, ServletException
    {
        res.setContentType("text/html");
        PrintWriter os=res.getWriter();
        os.println("<html>");
        os.println("<head><title>Ola!</title></head>");
        os.println("<h1 align=center>Ola!</h1>");
        os.println("</body></html>");
        os.close();
    }
}
```

63

# Servlets

## Exemplo

O método `doGet()` recebe dois objetos: um da classe `HttpServletRequest` e outro da classe `HttpServletResponse`.

O `HttpServletRequest` é responsável pela comunicação do cliente para o servidor e o `HttpServletResponse` é responsável pela comunicação do servidor para o cliente.

Primeiramente é usado o método `setContentType()` para definir o tipo do conteúdo a ser enviado ao cliente. Esse método deve ser usado apenas uma vez e antes de se obter um objeto do tipo `PrintWriter` ou `ServletOutputStream` para a resposta.

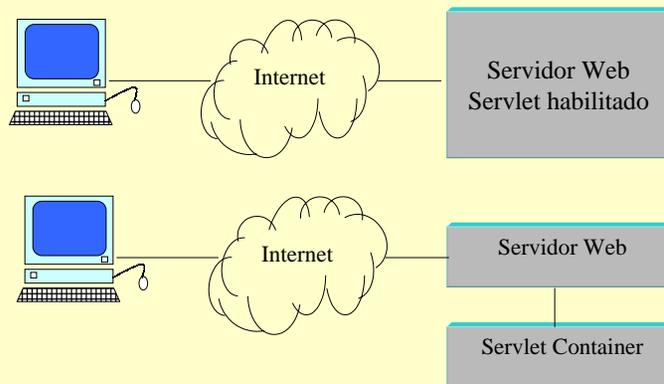
Se o programador desejar enviar a resposta em bytes deve usar o método `getOutputStream()` para obter um objeto `OutputStream`.

Uma vez carregado o Servlet não é mais descarregado, a não ser que o servidor Web tenha sua execução interrompida.

64

# Servlets

## Executando um servidor Habilitado para Servlet



65

# Servlets

## Executando o servidor Habilitado para Servlet

Assim como para se executar um Applet era preciso de um navegador Web com Java habilitado no caso de Servlets é preciso de servidor Web que execute Java ou que passe as requisições feitas a Servlets para programas que executem os Servlets.

66

# Servlets

## Executando um servidor Habilitado para Servlet

O primeiro caso é mais raro, tendo como exemplo o Java Web Server da Sun. O segundo caso é mais comum e existem *add-ons*, conhecidos como *Containers* ou *application Servers*, para os servidores mais populares como o Apache, IIS e o IBM HTTP Server.

Uma implementação de um *application Servers* capaz de lidar com Servlets e JSP pode ser encontrada em [jakarta.apache.org](http://jakarta.apache.org). Denominado de Tomcat, é um *container* que pode ser baixado gratuitamente, possui código aberto e possui *add-ons* para o Apache, IIS e Netscape Server.

67

# Servlets

## Instalando o Tomcat

Para instalar o tomcat basta descompactar o arquivo baixado na raiz do disco.

Após isso basta definir as seguintes variáveis de ambiente:

```
set TOMCAT_HOME=c:\jakarta-tomcat
set CLASSPATH=%CLASSPATH%;c:\jakarta-tomcat\lib\servlet.jar
```

Atualmente o Tomcat implementa as especificações Servlet 2.2 e JavaServer Pages 1.1

68

# Servlets

## Executando o Tomcat

Para iniciar o servidor basta executar o arquivo

```
c:\jakarta-tomcat\bin\startup.bat
```

Para interromper a execução servidor basta executar o arquivo

```
c:\jakarta-tomcat\bin\shutdown.bat
```

OBS: Caso ao iniciar o servidor apareça a mensagem `out of environment space` clique com o botão direito do mouse no arquivo `.bat` e edite as propriedades definindo o ambiente inicial com 4096.

69

# Servlets

## Executando o Tomcat

Ao entrar em execução o servidor lê as configurações do constantes no arquivo `server.xml` e por default se anexa à porta 8080

70

# Servlets

## Executando o Tomcat



71

# Servlets

## Executando o Servlet

Primeiro é preciso compilar o Servlet.

```
javac Ola.java
```

Coloque o arquivo gerado no diretório:

```
C:\jakarta-tomcat\webapps\examples\WEB-INF\classes
```

72

# Servlets

## Executando o Servlet

### Invocando diretamente pelo Navegador

Podemos executar um Servlet diretamente digitando a URL do Servlet no navegador. A URL em geral possui o seguinte formato:

```
http://máquina:porta/servlet/nome-servlet
```

Por exemplo, no caso do exemplo que estamos utilizando é preciso digitar:

```
http://localhost:8080/examples/servlet/Ola
```

O Servlet deve ser colocado em diretório predefinido ou sua localização relativa a esse diretório deve ser indicada na URL.

73

# Servlets

## Executando o Servlet

### Invocando em uma página HTML

No caso de uma página HTML basta colocar a URL nos locais normais de links. Por exemplo,

```
<a href="http://localhost:8080/examples/servlet/Ola">Servlet  
Ola</a>
```

Neste caso o Servlet Ola será solicitado quando o link associado ao texto "Servlet Ola" for acionado.

74

# Servlets

## Executando o Servlet

Note que a URL passada não existe. É uma URL virtual.

A primeira parte da URL (/examples) indica uma aplicação registrada no Tomcat.

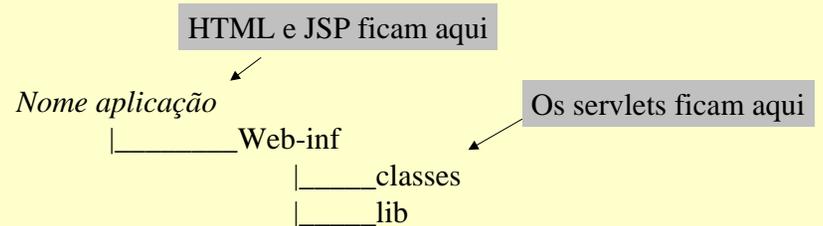
A segunda parte (/servlet) indica que é para ser executado um servlet.

75

# Servlets

## Criando uma aplicação no Tomcat

Crie a seguinte estrutura de diretórios abaixo do diretório webapps do Tomcat.



76

# Servlets

## Criando uma aplicação no Tomcat

É preciso também editar o arquivo `server.xml` do diretório `conf`, incluindo as linhas:

```
<Context path="/nome aplicação"
  docBase="webapps/ nome aplicação" debug="0"
  reloadable="true" >
</Context>
```

77

# Servlets

## Criando uma aplicação no Tomcat

E finalmente colocar uma arquivo `web.xml` no diretório `Web-inf` com o seguinte conteúdo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
<web-app>
</web-app>
```

78

# Servlets

## Exemplos de Servlets

79

# Servlets

## Exemplos Obtendo Informação sobre a Requisição

```
public class RequestInfo extends HttpServlet
{
  public void doGet(HttpServletRequest req,
    HttpServletResponse res)
    throws IOException, ServletException
  {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<html>\n<body>\n<head>");
    out.println("<title>Requisicao de Info </title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h3>Requisicao de Info </h3>");
    out.println("Metodo: " + req.getMethod()+"<br>");
    out.println("Request URI: " +
      req.getRequestURI()+"<br>");
    out.println("Protocolo: " + req.getProtocol()+"<br>");
  }
}
```

80

# Servlets

## Exemplos

### Obtendo Informação sobre a Requisição

```
out.println("PathInfo: " + req.getPathInfo()+"<br>");
out.println("Endereco remoto: " +
    req.getRemoteAddr()+"<br><br>");
Enumeration e = req.getHeaderNames();
while (e.hasMoreElements()) {
    String name = (String)e.nextElement();
    String value = req.getHeader(name);
    out.println(name + " = " + value+"<br>");
}
out.println("</body>");
out.println("</html>");
}

public void doPost(HttpServletRequest req,
    HttpServletResponse res)
    throws IOException, ServletException
{
    doGet(req, res);
}
}
```

81

# Servlets

## Saída

### Requisicao de Info

Metodo: GET  
Request URI: /servlet/RequestInfo  
Protocolo: HTTP/1.0  
PathInfo: null  
Endereco remoto: 127.0.0.1

Connection = Keep-Alive  
User-Agent = Mozilla/4.7 [en] (Win95; I)  
Pragma = no-cache  
Host = localhost:8080  
Accept = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, \*/\*  
Accept-Encoding = gzip  
Accept-Language = en  
Accept-Charset = iso-8859-1,\*,utf-8

82

# Servlets

## Exemplos

### Lidando com Formulários



83

# Servlets

## Exemplos

### Lidando com Formulários

```
public void doGet(HttpServletRequest req,
    HttpServletResponse res)
    throws IOException, ServletException
{
    res.setContentType("text/html");

    PrintWriter out = res.getWriter();
    out.println("<html>");
    out.println("<body>");
    out.println("<head>");
    out.println("<title>Trata formulario</title>");
    out.println("</head>");
    out.println("<body bgcolor=\"white\">");
    out.println("<h3>Trata formulario</h3>");
    String nome = req.getParameter("nome");
    String sobreNome = req.getParameter("sobrenome");
}
```

84

# Servlets

## Exemplos

### Lidando com Formulários

```
if (nome != null || sobreNome != null) {
    out.println("Nome = " + nome + "<br>");
    out.println("Sobrenome = " + sobreNome);
}
out.println("<P>");
out.print("<form action=\"");
out.print("Form\" ");
out.println("method=POST>");
out.println("Nome");
out.println("<input type=text size=20 name=nome>");
out.println("<br>");
out.println("Sobrenome");
out.println("<input type=text size=20
name=sobrenome>");
out.println("<br>");
out.println("<input type=submit>");
out.println("</form>");
```

85

# Servlets

## Exemplos

### Lidando com Formulários

```
out.println("</body>");
out.println("</html>");
out.close();
}

public void doPost(HttpServletRequest req,
                    HttpServletResponse res)
    throws IOException, ServletException
{
    doGet(req, res);
}
}
```

86

# Servlets

## Lidando com Cookies

Um *cookie* nada mais é que um pedaço de informação que é enviado do servidor para o navegador. O *cookie* é armazenado então no disco da máquina cliente e quando o site é novamente visitado o *cookie* enviado novamente para o servidor, fornecendo a informação desejada.

Os cookies são parte integrante do padrão Internet, normalizados pela norma RFC 2109.

87

# Servlets

## Lidando com Cookies



88

# Servlets

## Lidando com Cookies

```
public void doGet(HttpServletRequest req,
                 HttpServletResponse res)
    throws IOException, ServletException
{
    res.setContentType("text/html");

    PrintWriter out = res.getWriter();
    out.println("<html>");
    out.println("<body bgcolor=\"white\">");
    out.println("<head>");

    out.println("<title>Teste de Cookies</title>");
    out.println("</head>");
    out.println("<body>");

    out.println("<h3>Teste de Cookies</h3>");
}
```

89

# Servlets

## Lidando com Cookies

```
Cookie[] cookies = req.getCookies();
if (cookies.length > 0)
{
    for (int i = 0; i < cookies.length; i++)
    {
        Cookie cookie = cookies[i];
        out.print("Cookie Nome: " + cookie.getName() +
                "<br>");
        out.println("  Cookie Valor: " +
                cookie.getValue()+"<br><br>");
    }
}
```

90

# Servlets

## Lidando com Cookies

```
String cName = req.getParameter("cookienome");
String cValor = req.getParameter("cookievalor");
if (cName != null && cValor != null)
{
    Cookie cookie = new Cookie(cName ,cValor);
    res.addCookie(cookie);
    out.println("<P>");
    out.println("<br>");
    out.print("Nome : "+cName +"<br>");
    out.print("Valor : "+cValor);
}
}
```

91

# Servlets

## Lidando com Cookies

```
        out.println("<P>");
        out.print("<form action=\"");
        out.println("CookieTeste\" method=POST>");
        out.println("Nome");
        out.println("<input type=text length=20
name=cookienome><br>");
        out.println("Valor");
        out.println("<input type=text length=20
name=cookievalor><br>");
        out.println("<input type=submit></form>");

        out.println("</body>");
        out.println("</html>");
        out.close();
    }
}
```

92

# Servlets

## Lidando com Sessões

Não é possível implementar um *CE* sem acompanhar o usuário enquanto ele navega pelas páginas de um site, ou seja, é preciso ter um acompanhamento da sessão do usuário. No entanto, o protocolo HTTP é um protocolo não orientado à sessões. Como então implementar um acompanhamento de sessão na Web? Existem diversas forma de se fazer isso, mas a forma mais comum é acompanhar as sessões por meio de *cookies*.

93

# Servlets

## Lidando com Sessões

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse resp)
    throws IOException, ServletException
{
    resp.setContentType("text/html");

    PrintWriter out = resp.getWriter();
    out.println("<html>");
    out.println("<body bgcolor=\"white\">");
    out.println("<head>");

    out.println("<title>Teste de Sessao</title>");
    out.println("</head>");
    out.println("<body>");
```

94

# Servlets

## Lidando com Sessões

```
out.println("<h3>Teste de Sessao</h3>");
HttpSession session = req.getSession();
out.println("Identificador: " + session.getId());
out.println("<br> Data: ");
out.println(new Date(session.getCreationTime()));
out.println("<br>");
out.println("Ultimo acesso: ");
out.println(new Date(session.getLastAccessedTime()));

String nomedado = req.getParameter("nomedado");
String valordado = req.getParameter("valordado");
if (nomedado != null && valordado != null)
{
    session.putValue(nomedado, valordado);
}
```

95

# Servlets

## Lidando com Sessões

```
out.println("<P>");
out.println("Dados da Sessao: <br>");
String[] valueNames = session.getValueNames();
if (valueNames != null && valueNames.length > 0)
{
    for (int i = 0; i < valueNames.length; i++)
    {
        String name = valueNames[i];
        String value =
            session.getValue(name).toString();
        out.println(name + " = " + value + "<br>");
    }
}
```

96

# Servlets

---

## Lidando com Sessões

```
out.print("<P><form action=\"");
out.print("SessionExample\" ");
out.println("method=POST");
out.println("Nome");
out.println("<input type=text size=20 name=nomedado>");
out.println("<br>Valor");
out.println("<input type=text size=20 name=valordado>");
out.println("<br><input type=submit>");
out.println("</form>");
out.println("</body>");
out.println("</html>");
}
```

97

# JSP

98

# JSP

---

Servlets é uma boa idéia, mas você se imaginou montando uma página complexa usando println?

Devido a esse problema a Sun desenvolveu uma tecnologia baseada em Servlets chamada de JSP.

*Java Server Pages* (JSP) são páginas HTML que incluem código Java e outros *tags* especiais.

Desta forma as partes estáticas da página não precisam ser geradas por println. Elas são fixadas na própria página. A parte dinâmica é gerada pelo código JSP.

99

# JSP

---

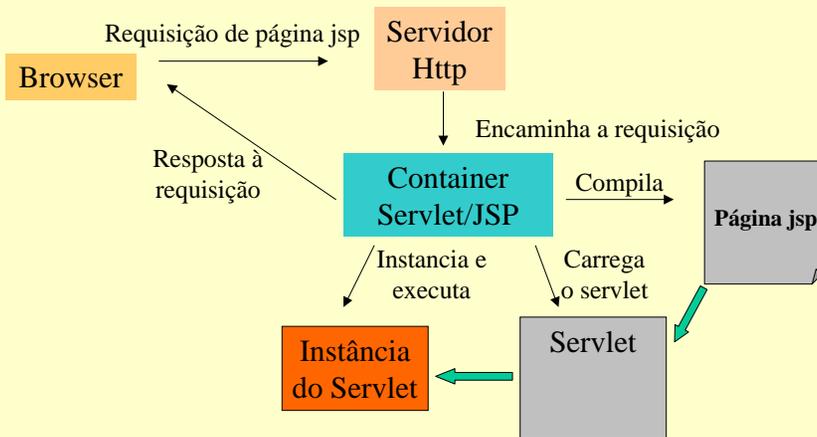
Assim a parte estática da página pode ser projetada por um artista que nada sabe de Java.

A primeira vez que uma página JSP é carregada pelo *container* JSP o código Java é compilado gerando um Servlet que é executado, gerando uma página HTML que é enviada para o navegador.

100

# JSP

## Como os dados são passados do cliente para servidor



101

# JSP

## Primeiro exemplo em JSP

```
<html><body>
<H1>O tempo em segundos e':
<%= System.currentTimeMillis()/1000 %>
</H1>
</body></html>
```

102

# JSP

## Executando o arquivo JSP

Para executar o arquivo acima salve-o com a extensão .jsp. Por exemplo tempo.jsp. Se você estiver usando o servidor Tomcat, crie um subdiretório a partir do diretório examples/jsp do Tomcat e salve o arquivo nesse diretório. Por exemplo examples/jsp/teste. Para invocar o arquivo JSP basta embutir a URL em uma página ou digitar diretamente a seguinte URL no navegador.

```
http://localhost:8080/examples/jsp/teste/tempo.jsp
```

O resultado será como abaixo, apenas com um número maior.

**O tempo em segundos e': 959622896**

103

# JSP

## Objetos implícitos

Para simplificar o código JSP existem um conjunto de objetos predefinidos que podem ser usados pelo programador. Os mais importantes são:

```
HttpServletRequest request
HttpServletResponse response
HttpSession session
PrintWriter out
```

104

# JSP

## Formato Básico

Os tags JSP possuem a seguinte forma geral:

```
<% Código JSP %>
```

O primeiro caractere % pode ser seguido de outros caracteres que determinam o significado preciso do código dentro do tag. Os tags JSP possuem correspondência com os tags XML.

105

# JSP

## Formato Básico

Expressões

```
<%= expressões %>
```

Expressões são avaliadas, convertidas para String e colocadas na página enviada. A avaliação é realizada em tempo de execução, quando a página é requisitada.

Exemplos:

```
<%= new java.util.Date() %>  
<%= request.getRemoteHost() %>
```

106

# JSP

## Formato Básico

Scriptlets

```
<% código java %>
```

Para tarefas mais complicada existem os Scriptlets permitem inserir trechos de código em Java.

Exemplo:

```
<%  
    String queryData = request.getQueryString();  
    out.println("Dados GET associado: " + queryData);  
%>
```

107

# JSP

## Formato Básico

Scriptlets

O código dentro do scriptlet é inserido da mesma forma que é escrito e todo o texto HTML estático antes e após ou um scriptlet é convertido para comandos print. Desta forma o scriptlets não precisa conter comandos para código estático e blocos de controle abertos afetam o código HTML envolvidos por scriptlets.

108

# JSP

## Formato Básico

Scriptlets

Exemplo:

```
Previs&atilde;o do Tempo
<% if (Math.random() < 0.5) { %>
Hoje vai <B>fazer sol</B>!
<% } else { %>
Hoje vai <B>chover</B>!
<% } %>
```

e convertido em

```
out.println("Previs&atilde;o do Tempo");
if (Math.random() < 0.5) {
    out.println(" Hoje vai <B>fazer sol</B>!");
} else {
    out.println(" Hoje vai <B>chover</B>!");
}
```

109

# JSP

## Formato Básico

Declarações

```
<%! Código Java %>
```

Uma declaração JSP permite definir variáveis e métodos que são inseridos no corpo do servlet.

Como as declarações não geram saída, elas são normalmente usadas em combinação com expressões e scriptlets.

110

# JSP

## Formato Básico

Declarações

Exemplo

O trecho abaixo imprime o número de vezes que a página corrente foi requisitada desde que foi carregada.

```
<%! private int numAcesso = 0; %>
Acessos desde carregada:
<%= ++ numAcesso %>.
```

111

# JSP

## Formato Básico

Declarações

As variáveis declaradas desta forma serão variáveis de instância. Já as variáveis declaradas em scriptlets são variáveis locais ao método `_jspService`.

112

# JSP

## Formato Básico

Diretivas

```
<%@ Diretiva atributo="valor" %>
```

ou

```
<%@ Diretiva atributo1="valor1"  
      atributo2="valor2"  
      ...  
      atributoN="valorN" %>
```

113

# JSP

## Formato Básico

Diretivas

Diretivas são mensagens para JSP container. Elas não enviam nada para a página mas são importantes para definir atributos JSP e dependências com o JSP container.

114

# JSP

## Formato Básico

Diretiva page

A diretiva page permite a definição dos seguintes atributos:

```
import="package.class" ou  
import="package.class1,...,package.classN"
```

Permite especificar os pacotes que devem ser importados.

Exemplo:

```
<%@ page import="java.util.*" %>
```

115

# JSP

## Formato Básico

Diretiva page

```
contentType="MIME-Type"
```

Especifica o tipo MIME da saída. O default é text/html.

Exemplo:

```
<%@ page contentType="text/plain" %>
```

possui o mesmo efeito do scriptlet

```
<% response.setContentType("text/plain"); %>
```

116

# JSP

## Formato Básico

Diretiva page

```
isThreadSafe="true|false"
```

Um valor true (default) indica um processamento normal do Servlet, onde múltiplas requisições são processadas simultaneamente.

Um valor false indica que o processamento deve ser feito por instancias separadas do Servlet ou serialmente.

117

# JSP

## Formato Básico

Diretiva page

```
session="true|false"
```

Um valor true (default) indica que a variável predefinida session (HttpSession) deve ser associada à sessão, se existir, caso contrário uma nova sessão deve ser criada e associada a ela.

Um valor false indica que nenhuma sessão será usada.

118

# JSP

## Formato Básico

Diretiva page

```
buffer="sizekb|none"
```

Especifica o tamanho do buffer para escrita.

```
autoflush="true|false"
```

Um valor true (default) indica que o buffer deve ser esvaziado quando estiver cheio.

```
info="message"
```

Define uma string que pode ser recuperada via `getServletInfo()`

119

# JSP

## Formato Básico

Diretiva page

```
errorPage="url"
```

Especifica a página JSP que deve ser processada em caso de exceções não capturadas.

```
isErrorPage="true|false"
```

Indica se a página corrente pode atuar como página de erro para outra página JSP. O default é false.

120

# JSP

## Formato Básico

Diretiva page

```
language=" java"
```

Possibilita definir a linguagem que está sendo usada. No momento a única possibilidade é Java.

121

# JSP

## Formato Básico

Diretiva include

```
<%@ include file="relative url" %>
```

Permite incluir arquivos na hora que a página JSP é traduzida em um servlet.

Exemplo:

```
<%@ include file="/meuarq.html" %>
```

122

# JSP

## Extraindo Valores de Formulários

```
<%@ page import="java.util.*" %>
<html><body>
<H1>Form</H1><H3>
<%
Enumeration flds = request.getParameterNames();
if(!flds.hasMoreElements()) { %>
  <form method="POST" action="Form.jsp">
  <% for(int i = 0; i < 10; i++) { %>
    Field<%=i%>: <input type="text" size="20"
      name="Field<%=i%>" value="Value<%=i%>"><br>
  <% } %>
  <INPUT TYPE=submit name=submit value="envie"></form>
<% } else {
  while(flds.hasMoreElements()) {
    String field = (String)flds.nextElement();
    String value = request.getParameter(field); %>
    <li><%= field %> = <%= value %></li>
  <% }
} %>
</H3></body></html>
```

123

# JSP

## Lidando com sessões

```
<html><body>
<H1>Session id: <%= session.getId() %></H1>
<H3><li>Essa sessão foi criada em
<%= session.getCreationTime() %></li></H3>
<H3><li>Antigo MaxInactiveInterval =
  <%= session.getMaxInactiveInterval() %></li>
<% session.setMaxInactiveInterval(5); %>
<li>Novo MaxInactiveInterval=
  <%= session.getMaxInactiveInterval() %></li>
</H3>
<H2>Se o objeto "teste" ainda existe este valor ser&acuteno nulo:<H2>
<H3><li>valor para "teste" =
  <%= session.getAttribute("teste") %></li></H3>
<% session.setAttribute("teste",
new String("alcione")); %>
<H1>Meu nome &acuteno;
<%= session.getAttribute("teste") %></H1>
```

124

# JSP

## Lidando com sessões

```
<FORM TYPE=POST ACTION=sessao2.jsp>
<INPUT TYPE=submit name=submit
Value="mantem"></FORM>
</body></html>
```

125

# JSP

## Lidando com sessões

```
<html><body>
<H1>Session id: <%= session.getId() %></H1>
<H1>Session value for "teste"
<%= session.getValue("teste") %></H1>
<FORM TYPE=POST ACTION=sessao.jsp>
<INPUT TYPE=submit name=submit Value="Returna">
</FORM>
</body></html>
```

126

# JSP

## Criando e Modificando Cookies

```
<html><body>
<H1>Session id: <%= session.getId() %></H1>
<%
Cookie[] cookies = request.getCookies();
for(int i = 0; i < cookies.length; i++) { %>
Cookie name: <%= cookies[i].getName() %> <br>
value: <%= cookies[i].getValue() %><br>
antiga idade máxima em segundos:
<%= cookies[i].getMaxAge() %><br>
<% cookies[i].setMaxAge(5); %>
nova idade máxima em segundos:
<%= cookies[i].getMaxAge() %><br>
<% } %>
<%! int count = 0; int dcount = 0; %>
<% response.addCookie(new Cookie(
"APO" + count++, "teste " + dcount++)); %>
</body></html>
```

127

# JSP

## Uso de javabeans

A medida que o código Java dentro do HTML torna-se cada vez mais complexo o desenvolvedor pode-se perguntar: Java em HTML não é o problema invertido do HTML em Servlet? Em outras palavras, estou novamente misturando conteúdo com forma?

Para solucionar esse problema é possível pode-se usar Javabeans para manipular a parte dinâmica em Java.

128

# JSP

---

## Uso de javabeans

Um Javabean nada mais é que uma classe Java que obedece a uma certa padronização de chamada.

Os atributos de um bean são acessados por meio de métodos que obedecem a convenção getXxxx e setXxxx. Exemplo: getItem()

129

# JSP

---

## Uso de javabeans

A sintaxe para o uso de um bean é:

```
<jsp:useBean id="name" class="package.class" />
```

Você também pode modificar o atributo scope para estabelecer o escopo do bean além da página corrente.

```
<jsp:useBean id="name" scope="session"
              class="package.class" />
```

130

# JSP

---

## Uso de javabeans

### Modificando os atributos:

Para modificar os atributos você pode usar o `jsp:setProperty` ou chamar um método explicitamente em um scriptlet.

131

# JSP

---

## Uso de javabeans

### Modificando os atributos:

Quando é dito que um bean tem uma propriedade prop do tipo T significa que o bean deve prover um método `getProp()` e um método do tipo `setProp(T)`.

132

# JSP

## Uso de javabeans

### Exemplo: página JSP

```
<HTML> <HEAD>
<TITLE>Uso de beans</TITLE>
</HEAD> <BODY> <CENTER>
<TABLE BORDER=5> <TR><TH CLASS="TITLE"> Uso de JavaBeans
</TABLE> </CENTER> <P>
<jsp:useBean id="test" class="curso. BeanSimples" />
<jsp:setProperty name="test" property="message" value="Ola mundo!"
/>
<H1>Message: <I>
<jsp:getProperty name="test" property="message" /> </I></H1>
</BODY> </HTML>
```

133

# JSP

## Uso de javabeans

### Exemplo: JavaBean curso/BeanSimples.java

```
package curso;

public class SimpleBean {
    private String message = "No message
specified";

    public String getMessage() {
        return(message);
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

134

# JSP

## Uso de javabeans

### scope

Existem quatro valores possíveis: page, request, session e application.

O default é page.

135

# JSP

## Uso de javabeans

### scope page

Objetos declarados com nesse escopo são válidos até a resposta ser enviada ou a requisição ser encaminhada para programa no mesmo ambiente, ou seja, só podem ser referenciados nas páginas onde forem declarados. Objetos declarados com escopo page são armazenados no objeto pagecontext..

136

# JSP

---

## Uso de `javabeans`

### scope request

Objetos declarados com nesse escopo são válidos durante a requisição e são acessíveis mesmo quando a requisição é encaminhada para programa no mesmo ambiente. Objetos declarados com escopo `request` são armazenados no objeto `request`.

137

# JSP

---

## Uso de `javabeans`

### scope session

Objetos declarados com nesse escopo são válidos durante a sessão desde que a página seja definida para funcionar em uma sessão. Objetos declarados com escopo `session` são armazenados no objeto `session`.

138

# JSP

---

## Uso de `javabeans`

### scope application

Objetos declarados com nesse escopo são acessíveis por páginas no mesmo servidor de aplicação. Objetos declarados com escopo `application` são armazenados no objeto `application`.

139

# JSP

---

## Uso de `javabeans`

### Importante

- Se no `setProperty` usarmos o valor “\*” significa que toda modificação em elementos do formulário será automaticamente passada para a propriedade do bean de mesmo nome.
- Os valores são automaticamente convertidos para o tipo correto no bean.

140

# JSP

## Uso de javabeans: carrinho de compras



141

# JSP

## Uso de javabeans

```
<html>
<jsp:useBean id="compra" scope="session"
  class="sessions.Compras" />
<jsp:setProperty name=" compra" property="*" />
<%
    compra.processRequest(request);
%>
<FONT size = 5 COLOR="#CC0000">
<br> voc&ecirc; comprou os seguintes itens:
<ol>
<%
    String[] items = compra.getItems();
    for (int i=0; i<items.length; i++) {
%>
<li> <%= items[i] %>
<%
    }
%>
</ol>
</FONT>
<hr>
```

142

# JSP

## Uso de javabeans

```
<body bgcolor="white">
<font size = 5 color="#CC0000">
<form type=POST action=compras.jsp>
<BR>Entre um item para adicionar ou remover:<br>
<SELECT NAME="item">
<OPTION>Televis&atilde;o
<OPTION>R&aacute;dio
<OPTION>Computador
<OPTION>V&iacute;deo Cassete
</SELECT>
<br> <br>
<INPUT TYPE=submit name="submit" value="adicione">
<INPUT TYPE=submit name="submit" value="remova">
</form></FONT></body></html>
```

143

# JSP

## Uso de javabeans

### O javabean Compras

```
package sessions;

import javax.servlet.http.*;
import java.util.Vector;
import java.util.Enumeration;

public class Compras {
    Vector v = new Vector();
    String submit = null;
    String item = null;

    private void addItem(String name) {
        v.addElement(name);
    }

    private void removeItem(String name) {
        v.removeElement(name);
    }
}
```

144

# JSP

## Uso de javabeans

```
public void setItem(String name) {
    item = name;
}

public void setSubmit(String s) {
    submit = s;
}

public String[] getItems() {
    String[] s = new String[v.size()];
    v.copyInto(s);
    return s;
}

private void reset() {
    submit = null;
    item = null;
}
```

145

# JSP

## Uso de javabeans

```
public void processRequest(HttpServletRequest request) {

    if (submit == null)
        return;

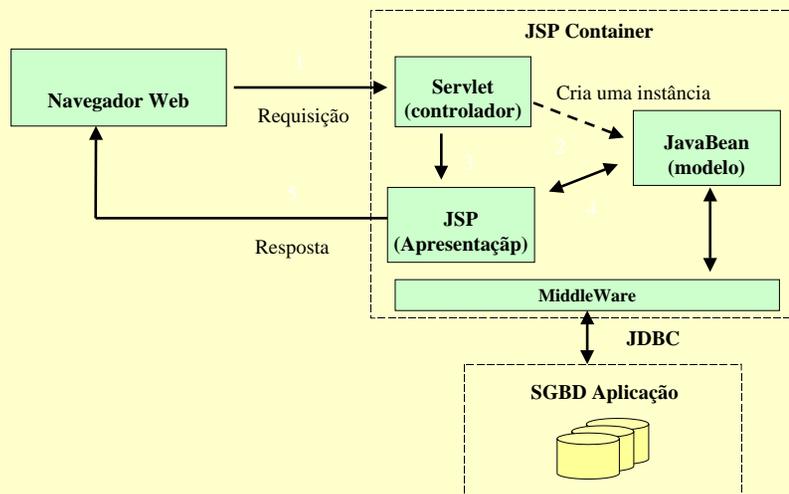
    if (submit.equals("adicione"))
        addItem(item);
    else if (submit.equals("remova"))
        removeItem(item);

    reset();
}
```

146

# JSP

## Uma Arquitetura para comércio eletrônico



147

# JSP

## Uma Arquitetura para comércio eletrônico

- Servlets - Atuam como controladores, recebendo as requisições dos usuários e acionando os beans e páginas JSP.
- JavaBeans - Atuam como modelo da solução, independente da requisição e da forma de apresentação. Comunicam-se com a camada intermediária que encapsula a lógica do problema.
- JSP - Atuam na camada de apresentação utilizando os javabeans para obtenção dos dados a serem exibidos, isolando-se assim de como os dados são obtidos.
- Middleware - Incorporam a lógica de acesso aos dados. Permitem isolar os outros módulos de problemas como estratégias de acesso aos dados e desempenho.

148



# JSP

## agenda: Exemplo de aplicação Web com arquitetura MVC

### Instalação

Os arquivos devem ser colocados nos seguintes diretórios do servidor:

#### Servlets e Beans (após compilados)

TOMCAT\_HOME/webapps/agenda/WEB-INF/classes/agenda

#### JSP e HTML

TOMCAT\_HOME/webapps/agenda

153

# JSP

## agenda: Exemplo de aplicação Web com arquitetura MVC

### Considerações sobre a solução

.O JavaBean da classe LoginBean é armazenado na sessão para permitir a verificação se o acesso ao site é autorizado.

.O JavaBean da classe AcaoBean é armazenado no objeto request uma vez que sua informações são alteradas a cada requisição.

154

# Ambientes para Servlet/JSP

- Visual Age for java 3.5 professional ou Enterprise (IBM)
- Forte (Sun)
- Jbuilder (Inprise)
- JDeveloper (Oracle)
- WebGain Studio, antigo Visual Café (WebGain)

155

# Bibliografia

- Conallen J.** *Building Web Applications With UML*, Addison-Wesley, 2000.
- Eckel B.** *Thinking in Java*. 2<sup>nd</sup> Ed. New Jersey : Prentice Hall, 2000.
- Wahli U. e outros.** *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*, IBM RedBooks, California, May 2000.
- Sadtler C. e outros.** *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*, IBM RedBooks, California, April 2000.

156

# Links

---

## Revistas

<http://www.javaworld.com/>

Revista online sobre Java.

## Livros

<http://www.eckelobjects.com/>

Página do autor do livro *Thinking in Java*, atualmente em segunda edição. O livro pode ser baixado gratuitamente no site.

<http://www.redbooks.ibm.com/booklist.html>

Livros da IBM

## Servidores

<http://jakarta.apache.org>

Página do projeto jakarta que desenvolveu o Tomcat.

<http://www.jboss.org>

Servidor de aplicação gratuito habilitado para EJB

157

# Links

---

## Site Java

<http://java.sun.com/>

## Site Oracle

<http://technet.oracle.com/>

## Site XML

<http://www.oasis-open.org/cover/xml.html>

<http://www.w3.org/>

## Site PHP

<http://www.php.net/>

## Site ASP

<http://aspbrasil.zip.net/>

## Site MySQL,

<http://www.mysql.com.br/>

## Outros

<http://www.weberdev.com/>

158

FIM

159