

- Esse modulo apresenta tudo que é necessário para implementar servlets escrevendo Java Server Pages
  - Sintaxe dos marcadores JSP e objetos
  - Funcionamento
  - Como implantar e depurar

- Tudo o que vale para servlets continua valendo para Java Server Pages
  - Um JSP é uma servlet durante a execução
  - Escrever código em JSP é como escrever código dentro do `doPost()` ou `doGet()` de um servlet com os objetos `request`, `response`, `out`, `session` e outros já definidos
  - Um JSP, depois de carregado, é tão veloz quando uma servlet (pois um JSP é uma servlet)
  - JSP é mais fácil de codificar e implantar, mas é mais difícil de depurar



# Problema das Servlets

---

- Servlets forçam o programador a embutir código HTML dentro de código Java
  - Desvantagem se a maior parte do que tem que ser gerado é texto ou HTML estático
  - Mistura as coisas, o programador tem que ser um bom Designer e se virar sem ferramentas gráficas



## O que são JSP's

---

- JSP é uma tecnologia padrão, baseada em templates para servlets. O mecanismo que a traduz é embutido no servidor
- Exemplo de um código em JSP

```
<body>  
<p>A data de hoje é <%=new Date() %>.</p>  
</body>
```



# O que são JSP's

---

- Em um servidor que suporta JSP, a pagina JSP é transformada (compilada) em uma servlet
- A forma mais simples de criar documentos JSP, é:
  - Mudar a extensão de um arquivo HTML para JSP
  - Colocar o documento em um servidor onde suporte JSP
- Fazendo isso, o servidor gera uma servlet
  - A compilação é feita no primeiro acesso
  - Nos acessos subseqüentes, a requisição é redirecionada a servlet gerada



# Exemplos JSP's

---

- Transformado em um JSP, um arquivo HTML pode conter :
  - Blocos de código (scriptlets) : `<% .... %>`
  - Expressões : `<%= ..... %>`

```
<p>Texto repetido:
```

```
<% for (int i = 0; i < 10; i++) { %>
```

```
    <p>Esta é a linha <%=i %>
```

```
<% }%>
```



# Ciclo de vida

---

- Quando uma requisição é mapeada para o JSP, o container :
  - Verifica se a servlet correspondente à pagina é mais antigo que a pagina (ou se não existe)
  - Se a servlet não existir, ou for mais antiga, a pagina JSP será compilada para gerar a nova servlet

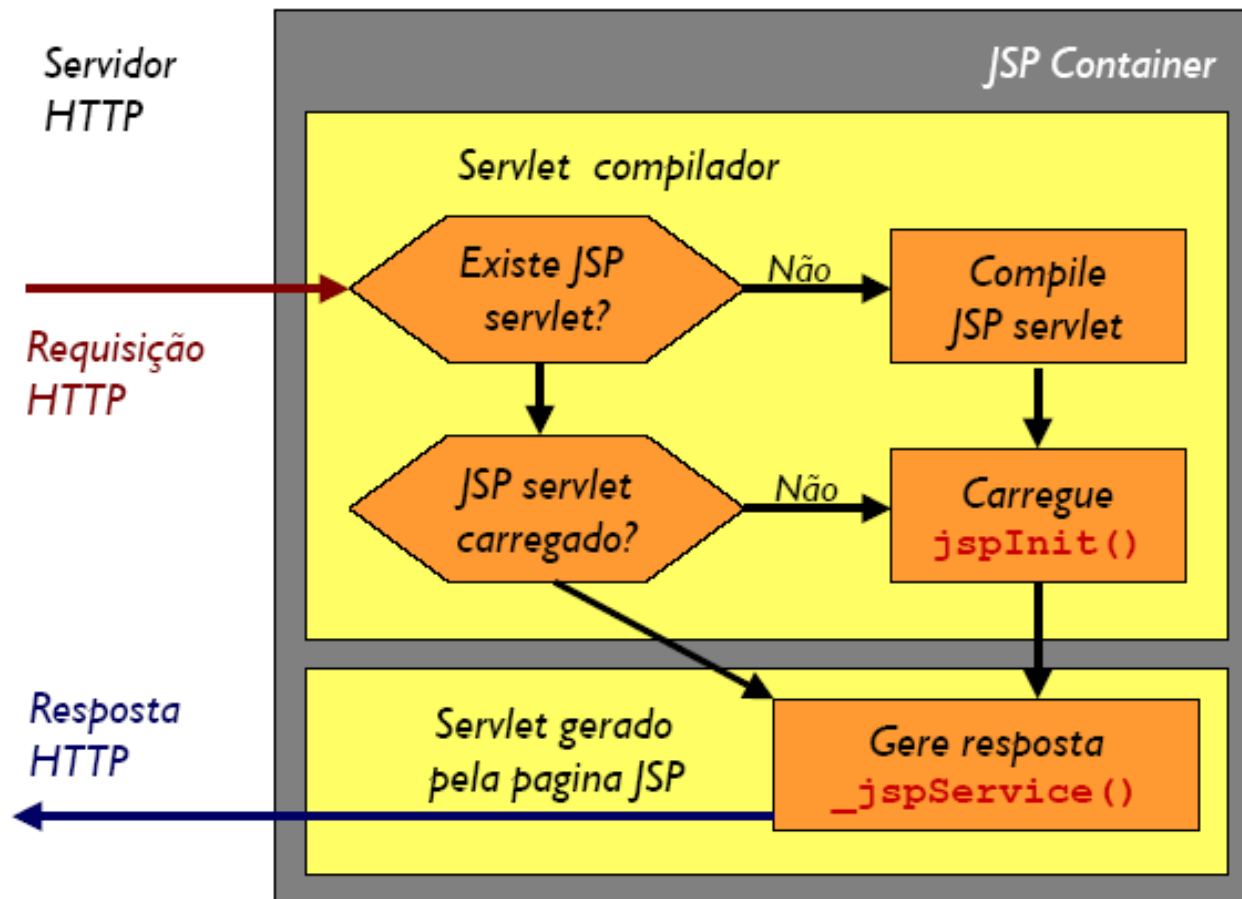


# Ciclo de vida

---

- Desse ponto em diante o ciclo de vida é similar ao das servlets, mas os métodos são diferentes
  - Se a servlet ainda não existir, ela será instanciada, e seu método `jspInit()` é chamado
  - Para cada requisição, seu método `_jspService(req,res)` é chamado
  - No fim da vida, o método `jspDestroy()` é chamado

# Como funciona uma JSP





# Sintaxe dos elementos da JSP

---

- Todos os elementos são interpretados no servidor (jamais chegam no navegador)
  - Diretivas: `<%@ ... %>`
  - Declarações: `<%! ... %>`
  - Expressões: `<%= ... %>`
  - Scriptlets: `<% ... %>`
  - Comentários: `<%-- .... --%>`
  - Ações: `<jsp:acao ... />`
  - Tags: `<prefixo:elemento ... />`

- Contém informações necessárias para gerar a servlet
- Sintaxe:
  - `<%@ diretiva atributo1 atributo2 .... %>`
- Principais diretivas:
  - `page` : atributos relacionados à página
  - `include` : inclui outros arquivos na página
  - `taglib` : declara biblioteca de custom tags usadas na página

- Exemplos:

```
<%@ page import="java.net.*, java.io.*"  
        session="false"  
        errorPage="/erro.jsp" %>  
<%@ include file="navbar.jsp" %>
```



# A diretiva page

---

- Atributos de `<%@ page %>`
  - `language="java"` (default)
  - `session="true"` (default)
  - `contentType="text/html; charset=ISSO-8859-1"` (default)
  - `extends="treinamento.FonteJSP"` (default : nenhum)
  - `import="java.util.*"` (default : java.lang.\*)
  - `autoFlush="true"` (default)
  - `isThreadSafe="true"` (default)
  - `errorPage="/erros/erro.jsp"` (default : nenhum)
  - `isErrorPage="false"` (default)



# A diretiva page

---

- session
  - Se true, aplicações JSP podem manter sessões do usuário abertas usando HttpSession
  - Se uma página declara false, ela não terá acesso a objetos gravados na sessão do usuário
- isThreadSafe
  - Se true, só um cliente poderá acessar a página ao mesmo tempo



# A diretiva page

---

- **isErrorPage**

- Se true, a página possui um objeto exception e pode extrair seus dados quando alvo de redirecionamento devido a erro.

- **errorPage**

- URL da página para o qual o controle será redirecionado na ocorrência de um erro ou exceção. Deve ser uma página com isErrorPage="true"



# Declarações

---

- Declarações dão acesso ao corpo da classe da servlet. Permitem a declaração de variáveis e métodos em uma JSP
- Úteis para declarar:
  - Variáveis e métodos de instância (pertencentes a servlet)
  - Variáveis e métodos estáticos (pertencentes a classe da servlet)
- Sintaxe:
  - `<%! Declaração %>`

- Exemplos

```
<%! public final static String[] meses =  
    {"jan", "fev", "mar", "abr", "mai", "jun"};
```

```
%>
```

```
<%! public static String getMes() {  
    Calendar cal = new GregorianCalendar();  
    return meses[cal.get(Calendar.MONTH)];  
}
```

```
%>
```



# Declarações (métodos especiais)

---

- `jspInit()` e `jspDestroy()` permitem maior controle sobre o ciclo de vida da servlet
  - Ambos são opcionais
  - Úteis para inicializar conexões, obter recursos, etc...
- Inicialização da página (chamado uma vez, antes da primeira requisição, após o instanciamento)

```
<%!  
    public void jspInit() { ... }  
%>
```



## Declarações (métodos especiais)

---

- Destruição da página (ocorre quando a servlet deixa a memória )

```
<%! public void jspDestroy() { ... } %>
```



# Expressões e scriptlets

---

- Expressões: Quando processadas, retornam um valor que é inserido na página
- Sintaxe:
  - `<%= expressão %>`
- Equivale a um `out.print(expressão)`, portanto não pode terminar em ponto-e-vírgula
  - Todos os valores resultantes das expressões são convertidos em String antes de serem colocados na tela



# Expressões e scriptlets

---

- Scriptlets: Blocos de código que são executados sempre que uma página JSP é processada
- Correspondem a inserção de seqüências de instruções no método `_jspService()`
- Sintaxe:
  - `<% instruções java ; %>`



# Comentários

---

- Comentários HTML `<!-- -->` não servem para comentar JSP
- `<!--` o texto ignorado pelo navegador, mas não pelo servidor `-->`
- Comentários JSP: podem ser usados para comentar blocos JSP
- `<%-- texto , código java , <HTML> ou tags <%JSP%>` ignorados pelo servidor `--%>`



# Ações padronizadas

---

- Sintaxe:

```
<jsp:nome_ação atrib1 atrib2 ... >  
  <jsp:param name="xxx" value="yyy"/>  
  ...  
</jsp:nome_ação>
```



# Ações padronizadas

---

- Permitem realizar operações externas ao servlet (tempo de execução)
  - Concatenação de várias páginas em uma única resposta
    - `<jsp:forward>` e `<jsp:include>`
  - Inclusão de JavaBeans
    - `<jsp:useBean>`, `<jsp:setProperty>` e `<jsp:getProperty>`



# Ações padronizadas (exemplos)

---

```
<%  
if (Integer.parseInt(totalImg) > 0) {  
%>  
    <jsp:forward page="selecimg.jsp">  
        <jsp:param name="totalImg"  
            value="<%= totalImg %>"/>  
        <jsp:param name="pagExibir" value="1"/>  
    </jsp:forward>  
%>  
} else {  
%>  
    <p>Nenhuma imagem foi encontrada.  
%>  
}  
%>
```



# Objetos implícitos JSP

---

- São variáveis locais previamente inicializadas
- Disponíveis nos blocos `<% ... %>` de qualquer página (exceto **session** e **exception**) que dependem de `@page` para serem ativados/desativados



# Objetos implícitos JSP

---

- **Objetos do servlet**
  - page
  - config
- **Objetos contextuais**
  - session
  - application
  - pageContext



# Objetos implícitos JSP

---

- Entrada e saída
  - request
  - response
  - out
- Controle de exceções
  - exception

- Referência para o servlet gerado pela página
  - Equivale ao "this" no servlet
- Pode ser usada para chamar qualquer método ou variável do servlet ou superclasses
- Exemplo:

```
<% HttpSession sessionCopy =  
    page.getSession() %>
```

- Referência para os parâmetros de inicialização da servlet (se existirem) através do objeto ServletConfig
- Equivale a `page.getSessionConfig()`
- Exemplo:

```
<% String user = config.getInitParameter("nome");  
String pass = config.getInitParameter("pass"); %>
```

- Parâmetros de inicialização são fornecidos na configuração do servlet no servidor, através de `<init-param>` em `<servlet>` no `web.xml`, é preciso declarar a página no `web.xml`

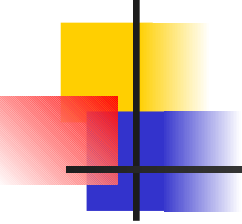
```
<servlet>
  <servlet-name>ServletJSP</servlet-name>
  <jsp-page>/pagina.jsp</jsp-page>
  <init-param>
    <param-name>nome</param-name>
    <param-value>guest</param-value>
  </init-param>
</servlet>
```

- Referência para os dados de entrada enviados na requisição do cliente (GET ou POST)
  - É um objeto do tipo `javax.servlet.http.HttpServletRequest`
- Usado para:
  - Guardar e recuperar atributos que serão usadas enquanto durar a requisição
  - Recuperar parâmetros passados pelo cliente
  - Descobrir o método usado (GET , POST)
    - `String método = request.getMethod();`

- URL no navegador:
  - <http://servidor/programa.jsp?nome=leandro&id=5>
- Recuperação dos parâmetros

```
<% String nome = request.getParameter("nome");
    String idStr = request.getParameter("id");
%>
<p> Bom dia <%=nome%>
```

- Referência aos dados de saída enviados na resposta
  - É um objeto do tipo `javax.servlet.http.HttpServletResponse`
- Usado para:
  - Definir o tipo de dados retornados (default: `text/html`)
  - Redirecionar
    - `response.sendRedirect("pagina2.html");`

- 
- Representa o stream de saída da página
    - É um objeto `javax.servlet.jsp.JspWriter`
  - Equivalente ao `response.getWriter()`
  - Principais métodos
    - `print()` e `println()` (imprimem Unicode)

- Os trechos abaixo são equivalentes

```
<% for (int i = 0; i < 10; i++) {  
  out.print("<p> Linha " + i);  
} %>
```

```
<% for (int i = 0; i < 10; i++) { %>  
<p> Linha <%= i %>  
<% } %>
```

- Representa a sessão do usuário
  - Uma instância de `javax.servlet.http.HttpSession`
- Útil para armazenar valores que deverão permanecer durante a sessão (`set/getAttribute()` )

```
Date d = new Date();  
session.setAttribute("hoje", d);
```

```
...
```

```
Date d = (Date)  
        session.getAttribute("hoje");
```

- Representa o contexto ao qual a página pertence
  - Instância de `javax.servlet.ServletContext`
- Útil para guardar valores que devem persistir pelo tempo que durar a aplicação (até que a servlet seja destruída pelo servidor )

```
Date d = new Date();  
application.setAttribute("hoje", d);  
...  
Date d = (Date)  
    application.getAttribute("hoje");
```

- Instância de `javax.servlet.jsp.PageContext`
- Oferece acesso a todos os outros objetos implícitos
  - `getPage()` retorna page
  - `getRequest()` retorna request
  - `getResponse()` retorna response
  - `getOut()` retorna out
  - `getSession()` retorna session
  - `getServletConfig()` retorna servletConfig
  - `getServletContext()` retorna application
  - `getException()` retorna exception

- Constrói a página (mesma resposta) com informações localizadas em outras URL's
  - `pageContext.forward(String)` similar a `<jsp:forward>`
  - `pageContext.include(String)` similar a `<jsp:include>`



# Escopo dos objetos

---

- A persistência das informações depende do escopo dos objetos onde elas estão disponíveis
- Constantes da classe PageContext identificam o escopo dos objetos

■ <i>pageContext</i>	<i>PageContext.PAGE_SCOPE</i>
■ <i>request</i>	<i>PageContext.REQUEST_SCOPE</i>
■ <i>session</i>	<i>PageContext.SESSION_SCOPE</i>
■ <i>application</i>	<i>PageContext.APPLICATION_SCOPE</i>



- Não existe em todas as páginas, apenas em páginas designadas como páginas de erro

```
<%@ page isErrorPage="true" %>
```

- Instância de `java.lang.Throwable`

- Exemplo:

```
<h1>Ocoreu um erro!</h1>
```

```
<p>A exceção é
```

```
<%= exception %>
```

```
Detalhes: <hr>
```

```
<% exception.printStackTrace(out) ; %>
```



# Exercícios

---

- Escreva um JSP `data.jsp` que imprima a data de hoje.

- Escreva um JSP temperatura.jsp que imprima uma tabela HTML de conversão Celsius-Fahrenheit entre -40 e 100 graus Celsius com incrementos de 10 em 10
  - A fórmula é  $F = 9/5 * C + 32$

- Altere o exercício anterior ,chame-o de `temperaturaRequest.jsp`, de forma que a tela apresente um campo texto para a entrada de um valor em celsius em um formulário que envie os dados como POST, a página deve mostrar:
  - O método pelo qual a página foi acessada
  - Mostrar na tela em vermelho o correspondente valor em Fahrenheit do valor passado em celsius

- Altere o exercício anterior ,chame-o de temperaturaErro.jsp, de forma que a tela redirecione qualquer exceção para uma tela de erro erro.jsp (que você deve criar)

- Crie um JSP chamado novaMensagem.jsp que contenha um formulário com o campo mensagem ,esse JSP deverá passar a mensagem por método POST para o JSP gravaMensagem.jsp ,o qual pegará a mensagem ,e a colocará em sessão.
- Crie um JSP chamado listaMensagem.jsp que liste as mensagens guardadas em sessão

- Crie um html que contenha um formulário com login e senha, e que submeta para uma servlet chamada `treinamento.exercicios.ServletLogin` usando método POST
- A `ServletLogin` deverá autenticar o login e senha baseado no arquivo `senhas.txt`, e redirecionar para o formulário caso o login seja inválido, para o JSP `compras.jsp` caso o perfil seja "usuario", ou para o JSP `adm.jsp` caso o perfil seja "adm".

- A tela adm.jsp deve permitir o cadastro de produtos na sessão
- A tela compras.jsp deve possuir componentes JSP de cabeçalho e rodapé, o rodapé deve conter o nome do usuário ,e o cabeçalho deve conter a lista de produtos cadastrados, essa tela deve permitir o usuário selecionar um produto, e adiciona-lo num carrinho de compras, e deve ter uma ação de efetuar compra, redirecionando para a tela lista.jsp, a qual deve mostrar os itens selecionados para a compra.