ITnerante

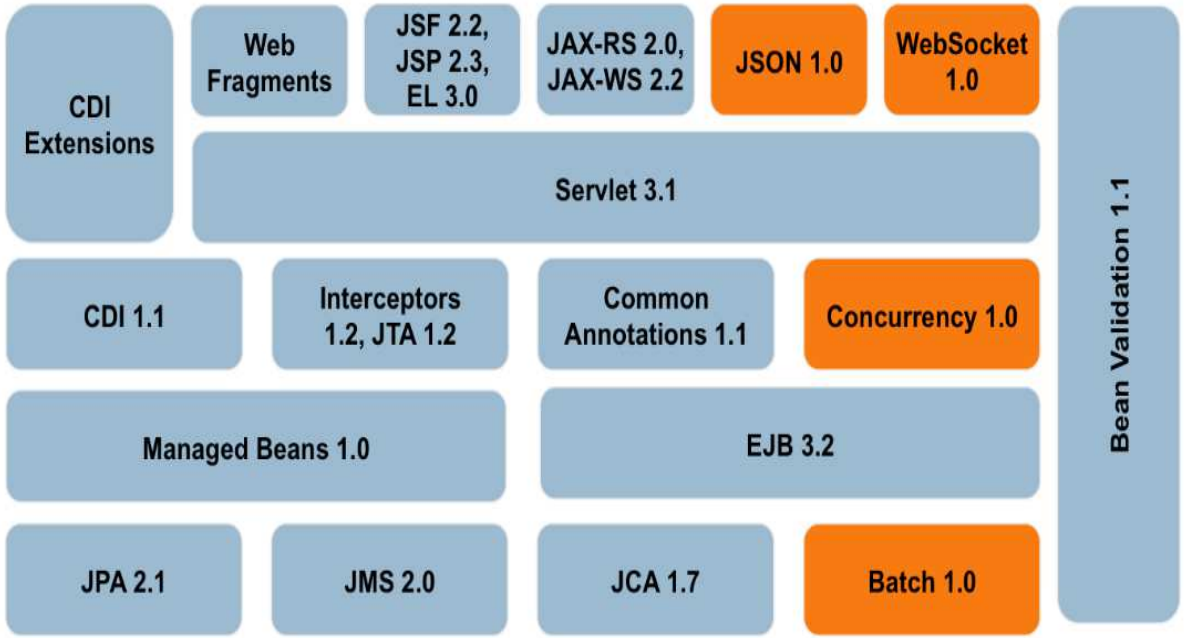
Java Enterprise Edition módulo III

PROVAS DE TI
TUDO PARA VOCÊ PASSAR

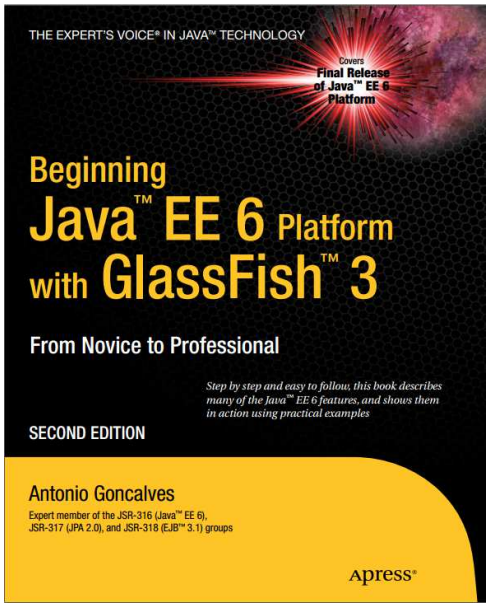
Leonardo Marcelino
http://www.itnerante.com.br/profile/LeonardoMarcelino

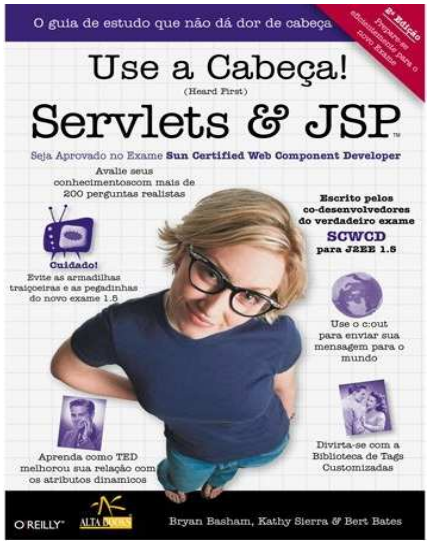
Java Enterprise Edition

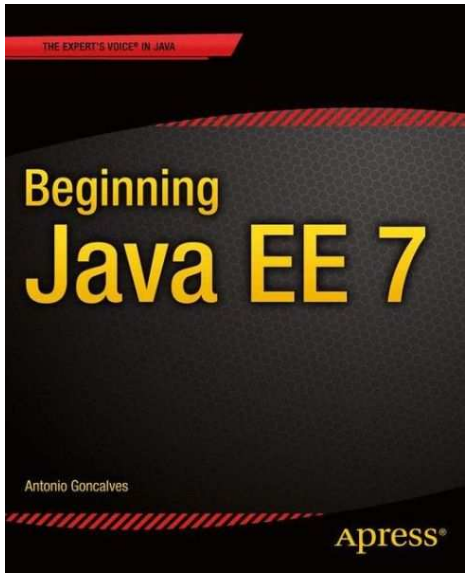
- Módulo 3
  - ✓Web Application Technologies
    - JavaServer Pages
    - EL
    - JSTL
    - JavaServer Faces
    - JSON-P
    - WebSocket

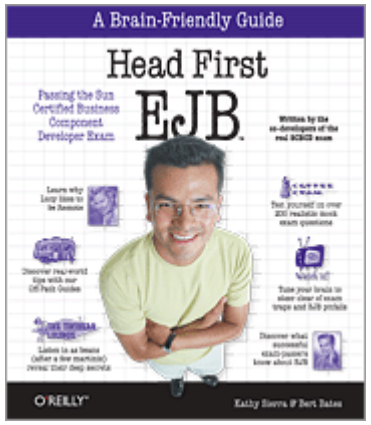


REFERÊNCIAS









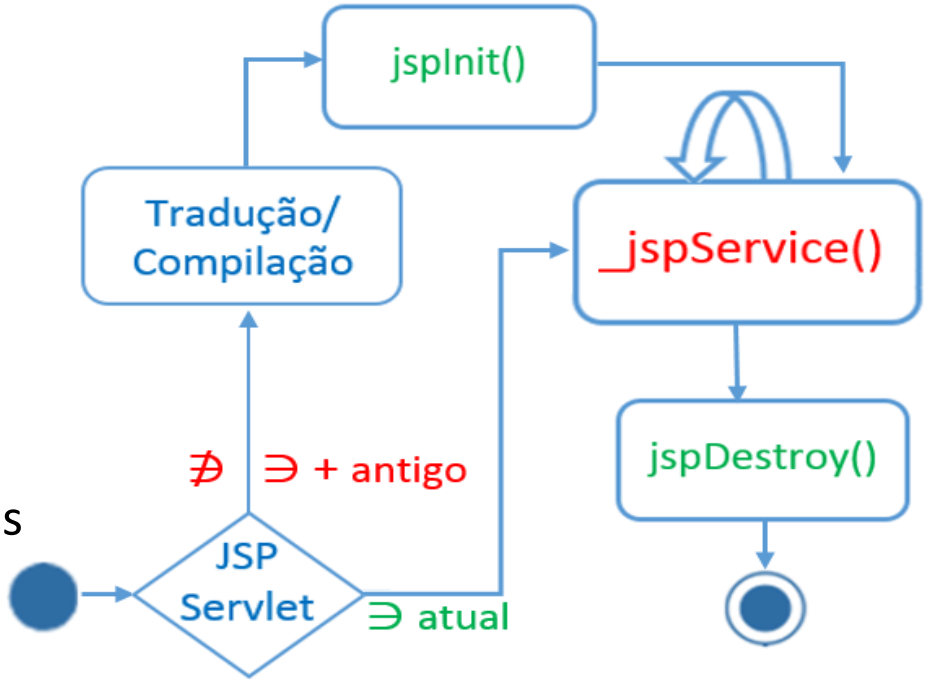
Java Server Pages

- roda no lado servidor
  - ✓ Componente que roda no container web
- Definição

  página JSP é um documento de texto que contém dois tipos de textos: dados estáticos e elementos JSP.
- Composição
  - ✓ Linguagem para desenvolver páginas JSP
  - ✓ Expression Language
    - ⇒ acessar objetos server-side
    - ⇒ JEE 6: EL 2.2 - JSR 245 (junto com JSP)
    - ⇒ JEE 7: EL 3.0 - JSR 341 (separado)
  - ✓ Mecanismos de extensão para JSP language
    - ⇒ custom tags
- Servlet x JSP
  - ✓ **Servlet**: código HTML na classe servlet
  - ✓ **JSP**: código java na página HTML
    - ⇒ elementos JSP (standard e XML)

JSP: Ciclo de Vida

- ✓ Semelhante ao servlet
- ✓ Gerenciado pelo container web



- ▷ Especificação
  - ⇒ Tradução
    - ☑ localiza ou cria a classe Servlet
      - ‣ JSP page implementation class
      - ‣ a partir dos elementos JSP
    - ☑ tradução ocorre quando:
      - ‣ JSP servlet não existe ou é mais antiga que JSP de origem
    - ☑ ciclo de vida do servlet (init)
  - ⇒ Execução
    - ☑ container gerencia instâncias do JSP Servlet
- ▷ Ciclo de vida do servletJSP
  - ☑ interface **`javax.servlet.jsp.JspPage`**
    - ‣ `jspInit()` e `jspDestroy()`

  - ☑ interface **`javax.servlet.jsp.HttpJspPage`**
    - ‣ `_jspService(HttpServletRequest, HttpServletResponse)`

[01]

CESPE - 2010 - TRT 10

Nas páginas JSP, combinam-se modelos estáticos, incluindo fragmentos de HTML ou XML, com o código para gerar conteúdo dinâmico e compilar páginas JSP dinamicamente em *servlets*, quando solicitado.

CESPE - 2011 - MEC

A JSP é uma extensão da tecnologia dos servlets que permite simplificar o processo de criação de páginas, separando a apresentação do conteúdo.

[02] FCC - 2012 - TRE-CE

Embora as servlets sejam muito boas no que fazem, tornou-se difícil responder ao cliente com conteúdo no formato HTML.

PORQUE

Geralmente quem trabalha com o conteúdo HTML é o web designer que normalmente não é programador Java experiente. Ao misturar HTML dentro de uma servlet, torna-se muito difícil separar as funções de web designer e desenvolvedor Java. Além disso, é difícil fazer alterações no conteúdo HTML, pois para cada mudança, uma recompilação da servlet tem que acontecer. Para contornar as limitações da tecnologia Java Servlet a Sun Microsystems criou a tecnologia JavaServer Pages (JSP).

Acerca dessas asserções, é correto afirmar:

a) Tanto a primeira quanto a segunda asserções são proposições falsas.
b) A primeira asserção é uma proposição verdadeira e a segunda uma proposição falsa.
c) A primeira asserção é uma proposição falsa e a segunda uma proposição verdadeira.
d) As duas asserções são proposições verdadeiras, mas a segunda não é a justificativa correta da primeira.
e) As duas asserções são proposições verdadeiras e a segunda é a justificativa correta da primeira.

[03] CESPE - 2004 - SERPRO

Cada JSP passa por duas fases distintas. Na primeira, denominada translation time, o tradutor transforma um arquivo em um servlet. A segunda fase ocorre quando o servlet é executado para a geração da página. A manipulação dos comentários ocorre na primeira fase, o tradutor omite qualquer comentário fazendo que o servlet não o tenha que manipular.

[04] CESPE - 2006 - CENSIPAM

O seguinte cenário descreve os passos que são executados toda vez que uma página JSP é solicitada: o navegador solicita a página JSP ao servidor; o código de um servlet é gerado e compilado; nesse servlet, o HTML da página encontra-se convertido em enunciados **println**; o servlet é instanciado e são invocados os métodos **init** e **service**; o servlet recebe dados sobre a solicitação via uma instância de **ServletRequest** e envia dados via uma instância de **ServletResponse**; a página HTML produzida pelo servlet é enviada para o navegador; a página é apresentada pelo navegador.

JSP: Elementos

- ✓ código dinâmico
- ✓ lógica da apresentação/negócio
- ✓ permite acessar objetos no servidor
- ✓ tipos de elementos
  - scripting, diretivas e ações (actions)
- ✓ dois tipos de sintaxe
  - ◇ **Sintaxe padrão**
    - elementos JSP scripting
    - código Java nas páginas JSP
    - HTML com extensão JSP
  - ◇ **XML document**
    - sintaxe XML
      - deve ser bem formado
    - XHTML com extensão JSP

❑ Elementos de scripting

	padrão	XML
scriptlets	<% %>	<jsp:scriptlet>
expressões	<%= %>	<jsp:expression>
declarações	<%! %>	<jsp:declaration>
comentários	<%-- --%>	<!-- -->

JSP: Elementos - Scripting

```jsp
<Html>
    <Head>
        <title>Elementos JSP</title>
    </Head>
    <Body>
        <!-- comentário HTML -->
        <%-- comentário JSP --%>
        <% /*comentário Servlet */  %>

        <%! String name = "Carlos"; %>
        <%! int countGeral = 0; %>

        <% int countLocal = 0; %>

        <h1> Bem vindo <%=name%></h1>

        <% ++countLocal;
           ++countGeral; %>

        <% if (countLocal > 4) { %>
            <h3> Teste OK</h3>
        <% } else { %>
            <h3> contador menor que 5</h3>
        <% } %>

        <% if (countGeral > 4) { %>
            <h3> Teste OK</h3>
        <% } else { %>
            <h3> contador menor que 5</h3>
        <% } %>

        <% out.println("Nome:" + name); %>
    </Body>
</Html>
```

```java
public class Elements_jsp extends HttpJspBase{

    int countGeral = 0;
    String name = "Carlos"; int age = 27;

    public void _jspService(HttpServletRequest request,
            HttpServletResponse response)
            throws java.io.IOException,ServletException {

        out.write("<html><body>");

        out.write("<!-- comentário HTML -->");
        /*comentário Servlet */

        out.write("<h1> Bem vindo ");
        out.print(name);
        out.write("</h1>");

        int countLocal= 0;

        ++countLocal;   ++countGeral;

        if (countLocal > 4)
            out.write("<h3>Teste OK</h3>");
        else
            out.write("<h3>contador menor que 5</h3>");

        out.println("Nome:" + name);
        out.write("</body></html>");
    }
```

JSP: Elementos - Scripting

```
<Html>
    <Head>
        <title>Elementos JSP</title>
    </Head>
    <Body>
        <!-- comentário HTML -->
        <%-- comentário JSP --%>
        <% /*comentário Servlet */  %>

        <%! String name = "Carlos"; %>
        <%! int countGeral = 0; %>

        <% int countLocal = 0; %>

        <h1> Bem vindo <%=name%></h1>

        <% ++countLocal;
           ++countGeral; %>

        <% if (countLocal > 4) { %>
            <h3> Teste OK</h3>
        <% } else { %>
            <h3> contador menor que 5</h3>
        <% } %>

        <% if (countGeral > 4) { %>
            <h3> Teste OK</h3>
        <% } else { %>
            <h3> contador menor que 5</h3>
        <% } %>

        <% out.println("Nome:" + name); %>
    </Body>
</Html>
```

```
<jsp:declaration>
    String name = "Carlos";
    int countGeral = 0;
</jsp:declaration>

<jsp:scriptlet>  int countLocal = 0; </jsp:scriptlet>

<h1> Bem vindo <jsp:expression> name  </jsp:expression> </h1>

<jsp:scriptlet>
    ++countLocal;
    ++countGeral;
</jsp:scriptlet>

<jsp:scriptlet> if (countLocal > 4) { </jsp:scriptlet>
        <h3>Teste OK</h3>
<jsp:scriptlet> } else { </jsp:scriptlet>
        <h3>contador menor que 5</h3>
<jsp:scriptlet> } </jsp:scriptlet>

<jsp:scriptlet> if (countGeral > 4) { </jsp:scriptlet>
        <h3>Teste OK</h3>
<jsp:scriptlet> } else { </jsp:scriptlet>
        <h3>contador menor que 5</h3>
<jsp:scriptlet> } </jsp:scriptlet>

<jsp:scriptlet> out.println("Nome:" + name); </jsp:scriptlet>
```

[05] FGV - 2012 - SENADO

Assinale a alternativa que apresenta instruções corretas em sintaxe válida JSP para inicializar e imprimir o valor da variável *num*.

a)
```
<%int num = 2;%>
<%=num; %>
```

b)
```
<%int num = 2
num; %>
```

c)
```
<%int num = 2 %>
<%=num %>
```

d)
```
<%int num = 2; %>
<%=num %>
```

e)
```
<%int num = 2 %>
<%=num; %>
```

[06]

CESPE - 2010 - MPU

O contêiner, que executa JSP, transforma o programa JSP em Servlet, assim, a expressão "<%= Math.Random()%>" se torna argumento para out.println().

CESPE - 2007 - CBM-DF

Uma página JSP pode conter métodos Java definidos entre os grupos de caracteres <%! e %>. Esses métodos não podem ser invocados a partir de expressões Java definidas entre <%= e %>. O resultado da execução de uma expressão é convertido em um int e incluído na página apresentada.

[07]

CESPE - 2011 - MEC

A JSP possui quatro componentes: chaves diretivas, ações, elementos de script e bibliotecas de tags. As ações são mensagens para o contêiner de JSP e os elementos de script permitem aos navegadores inserir códigos Java que interajam com os componentes JSP.

CESPE - 2013 - CNJ

Como forma de incluir dinamismo em páginas JSP, é possível incluir blocos de código Java conhecidos como *scriptlets*.

[08] FUMARC - 2011 - BDMG

Analise as seguintes afirmativas:

I. Uma página JSP é um documento texto que contém dados estáticos em formato HTML e XML, por exemplo, e elementos JSP que constroem o conteúdo dinâmico.

II. Os elementos JSP em uma página JSP podem ser expressos nas sintaxes padrão e XML, embora em um dado arquivo, pode-se usar somente uma das sintaxes.

III. Uma página JSP em sintaxe XML é um documento XML que pode ser manipulado pelas ferramentas e APIs para documentos XML. Além disso, pode ter um **jsp:root** como elemento raiz.

Marque a alternativa **CORRETA:**

a) apenas a I e II são verdadeiras.
b) apenas a I e III são verdadeiras.
c) apenas a II e III são verdadeiras.
d) todas são verdadeiras.

JSP: Elementos - Diretivas

- ✓ fornece instruções ao container
- ✓ afeta a estrutura global do JSP servlet
- ✓ não gera código ou saída na servlet
- ✓ processadas em tempo de tradução
- ✓ sintaxe
  - Standard: `<%@ directive attr1="v1" attr2="v2" %>`
  - XML: `<jsp:directive.type attribute="value}" />`
- ▷ **Tipos**
  - page: atributos relacionados à página
  - include: inclui recursos na página
    - `file="URL relativa"`

```
<%@ include file="URL_relativo" %>
<jsp:directive.include file="URL_rel" />
```

  - taglib: declara bibliotecas de tags usadas na página
    - `uri="tagLibraryURI"`
    - `tagdir="tagFilesDir"`
    - `prefix="tagPrefix"`

```
<%@ taglib uri|tagdir="uri" prefix="prefix" >
<some:tag xmlns:prefix="uri">
```

JSP: Elementos - Diretivas

- **Diretiva taglib**
  - Tag Files (JSP 2.0)
    - ✓ alternativa para bibliotecas de tags
    - ✓ arquivo com o handler de tag customizada
      - JSP com extensão .tag ou .tagx
    - ✓ localização
      - JAR em WEB-INF/lib
        - tagfiles em META-INF/tags
        - requer TLD com <tag-file>
      - WEB-INF/tags
        - TLD implícito gerado pelo container
        - valor do atributo `tagdir`
    - ✓ diretivas
      - include e taglib
      - exclusivas
        - tag, attribute e variable

```
<%@ tag display-name="formattedDate"
    import="java.text.SimpleDateFormat"
    import="java.util.Date"%>

<%@ attribute name="format"%>

<%  SimpleDateFormat formatter = null;
    if (format == null)
        formatter = new SimpleDateFormat();
    else
        formatter = new SimpleDateFormat(format);
%>

<%= formatter.format(new java.util.Date()) %>
```

```
<%@ taglib tagdir="/WEB-INF/tags" prefix="tf" %>

<html>
    <head>
    <title>Data Demo</title>
    <body>
        A data é: <tf:formattedDate format="d MMM YYYY" />
    </body>
</html>
```

JSP: Elementos - Diretivas

- **Diretiva page** `<%@ page` `<jsp:directive.page attribute="value" ... />`
  - ✓ define propriedades específicas da página JSP
  - ✓ especificação não permite duplicidade de atributos
    - exceto: import e pageEncoding
  - atributos
    - language
      - define a linguagem de scripting usada nos scriptlets, expressões e declarações
      - default: java
    - extends
      - define uma superclasse para a classe JSP Servlet
      - restringe a capacidade do container fornecer superclasses com melhor desempenho
    - import
      - especifica pacotes ou classes que serão adicionadas ao JSP Servlet
      - default: java.lang.*, javax.servlet.*, javax.servlet.jsp.* e javax.servlet.http.*
    - session
      - indica se o JSP Servlet terá um objeto implícito session.
      - default: true

JSP: Elementos - Diretivas

- **Diretiva page** `<%@ page` `<jsp:directive.page attribute="value" ... />`
  - buffer
    - define como o buffering é tratado pelo objeto implícito out (JspWriter)
    - se buffer="none" a saída vai direto para o stream
    - default: não menos que 8kb
  - autoFlush
    - define se o buffer deve ser esvaziado automaticamente
    - se false, uma exceção deve ser lançada para indicar buffer overflow
    - default: true
  - isThreadSafe
    - indica o nível de thread safe implementado na página JSP
    - se true, indica que o JSP Servlet é capaz de responder a múltiplas solicitações simultâneas (várias threads)
    - se false, o JSP Servlet reponde a apenas uma solicitação por vez
    - SingleThreadModel: forma mais comum de implementar o ThreadSafe (deprecated Servlet 2.4)
    - default: true
  - info
    - define uma String que pode ser recuperada pelo método getServletInfo()

JSP: Elementos - Diretivas

- **Diretiva page** `<%@ page` `<jsp:directive.page attribute="value" ... />`
  - isErrorPage
    - indica se a página JSP controla exceções
    - se true, as páginas JSP têm acesso ao objeto implícito exception
    - é referenciada pelo atributo errorPage de outros JSP
    - default: false
  - errorPage
    - define uma URL para página alternativa a ser exibida se um erro (não tratado) ocorrer
    - a página JSP da URL terá um atributo isErrorPage="true"
  - contentType
    - define o tipo MIME e o character encoding para a resposta JSP
    - default: text/html (sintaxe standard) ou text/xml (sintaxe XML Document)
  - pageEncoding
    - define o character encoding (charset) para o JSP.
    - default: ISO-8859-1 (sintaxe standard)
  - isELIgnored
    - define se as expressões EL serão ignoradas quando a página for traduzida
    - default: depende da versão do web.xml

[09] FCC - 2012 - MPE-PE

Em uma página JSP, para importar uma classe de um pacote e para fazer referência a uma biblioteca (como, por exemplo, JSTL) podem ser utilizadas, respectivamente, as diretivas:

a) `<%@page import="pacote.Classe"%> e <%@taglib uri="caminho/biblioteca" prefix="prefixo"%>`

b) `<%@include import= "pacote.Classe"%> e <%@taglib uri="caminho/biblioteca"%>`

c) `<%import= "pacote.Classe"%> e <%taglib uri="caminho/biblioteca"%>`

d) `<%@page include= "pacote.Classe"%> e <%@library uri="caminho/biblioteca"%>`

e) `<%@import class= "pacote.Classe"%> e <%@taglib urli="caminho/biblioteca"%>`

[10] FCC - 2012 - TRE-CE

<%@ page atributo1="valor1" atributo2="valor2"... %> é a sintaxe típica da diretiva Page, em JSP. Um de seus atributos, se definido para true, indica o processamento normal do servlet quando múltiplas requisições podem ser acessadas simultaneamente na mesma instância de servlet. Trata-se do atributo

a) extends.
b) import.
c) isThreadSafe.
d) session.
e) autoFlush.

JSP: Elementos - Ações

- ✓ fornece informações para tratar a request
- ✓ controlam o comportamento da engine de Servlet
- ✓ utilizam a sintaxe XML
- ✓ cria ou modifica objetos na página JSP
- ✓ sintaxe
  - `<jsp:action_name attribute="value" />`

▷ **Standard Actions**

jsp:useBean	jsp:param	jsp:invoque	jsp:output
jsp:setProperty	jsp:fallback	jsp:doBody	jsp:root
jsp:getProperty	jsp:text	jsp:element	jsp:declaration
jsp:include	jsp:plugin	jsp:body	jsp:scriptlet
jsp:forward	jsp:params	jsp:attribute	jsp:expression

JSP: Elementos - Ações

- <jsp:useBean>
  - ✓ simplifica instanciação/localização de objetos
    - Java Beans (POJOs)
  - ✓ atributos
    - id="objetoID"
    - class="Nomeclasse" ou beanName="nomeBean"
    - type="class|superclass|interface"
    - scope="**page**|request|session|application"

```
<% Person p = (Person) request.getAttribute("person");%>
Pessoa é: <%=p.getName()%>

<jsp:useBean id="person" class="beanPackage.Person" scope="request" />
<jsp:useBean id="supervisor" type="beanPackage.Funcionario" class="beanPackage.Supervisor" />

Pessoa é: <jsp:getProperty name="person" property="name" />
Pessoa é: ${person.name}

<jsp:setProperty name="person" property="name" value="Fred" />
${person.name("Fred")}

<jsp:useBean id="person2" class="beanPackage.Person" scope="request">
    <jsp:setProperty name="person2" property="name" value="Fred" />
</jsp:useBean>

<jsp:setProperty name="person2" property="name" param="userName" />
```

JSP: Elementos - Ações

- **<jsp:include>**
  - ✓ transferência temporária do controle para outro recurso
  - ✓ diretiva include `<%@ include file="Header.jsp" %>`
    - mesmo resultado na tela
    - mecanismo interno diferente
      - ⇒ diretiva: conteúdo escrito no JSP em out.write (translation-time)
      - ⇒ action: "chama" servlet correspondente ao recurso (runtime)
  - ✓ semelhante a RequestDispatcher.include
    - passa objetos request e response internamente
  - ✓ pode ter <jsp:param> como subelemento
  - ✓ sintaxe: `<jsp:include page="Header.jsp" flush="false"/>`
- **<jsp:forward>**
  - ✓ transferência permanente do controle para outro recurso
  - ✓ semelhante a RequestDispatcher.forward
    - mesmo comportamento para IllegalStateException
  - ✓ sintaxe:

```
<jsp:forward page="/TestServlet">
    <jsp:param value="sort" name="action" />
</jsp:forward>
```

JSP: Objetos implícitos

- ✓ instanciados na carga do método _jspService()
- ✓ referenciam recursos da API Servlet/JSP

▷ Objetos implícitos **JavaServer Pages**

- ✓ disponíveis para uso em scriptlets e expressões

Disponíveis em páginas JSP		
application	ServletContext	Application
session	HttpSession	Session
request	ServletRequest	Request
config	javax.servlet.ServletConfig	Page
out	javax.servlet.jsp.JspWriter	
page	java.lang.Object	
pageContext	javax.servlet.jsp.PageContext	
response	ServletResponse	

Disponíveis em páginas de erro		
exception	java.lang.Throwable	Page

Alternativa

Servlet da página

page config

Output e input da página

request response

Contextuais

application session

Request pageContext

Erros

exception

[11] ESAF - 2009 - ANA

O mecanismo de inclusão, que permite o conteúdo dinâmico ser incluído em uma JSP em tempo de solicitação, é denominado:

a) Ação <jsp:plugin>.

b) Ação <jsp:include>.

c) Diretiva include.

d) Diretiva Page.

e) Diretiva taglib.

[12] FCC - 2013 - ALE-RN

Em uma aplicação web desenvolvida utilizando a plataforma Java EE 6, há a seguinte classe Java:

```
package dados;
public class Cliente {
        private String nome;
        public Cliente() { }
        public String getNome() {
                return nome;
        }
        public void setNome(String nome) {
                this.nome = nome;
        }
}
```

Em uma página JSP da mesma aplicação, para instanciar um objeto desta classe pode-se utilizar a tag

a) <jsp:setBean name="cliente" class="dados.Cliente"/>

b) <jsp:setBean id="cliente" class="dados.Cliente"/>

c) <jsp:useBean name="cliente" class="dados.Cliente"/>

d) <jsp:useBean id="cliente" class="dados.Cliente"/>

e) <jsp:newInstance id="cliente" class= "dados.Cliente"/>

[13] CESPE - 2008 - MPE-RR

Cada vez que um novo pedido http for enviado por um *browser* a essa página, será criada uma nova instância da classe sessions.DummyCart.

```
<html>
   <jsp:useBean id="cart" scope="session" class="sessions.DummyCart" />
   <jsp:setProperty name="cart" property="*" />
   <%
      cart.processRequest(request);
   %>
   <FONT size=5 color="#CC0000">
      <br> You have the following items in your cart:
      <ol>
         <%
            String[] items = cart.getItems();
            for (int i=0; i<items.length; i++) {
            %>
              <li><% out.print(util.HTMLFilter.filter(items[i])); %>
            <%
            }
         %>
      </ol>
   </FONT>
   <hr>
   <%@ include file ="carts.html" %>
</html>
```

[14]

CESPE - 2011 - PREVIC

O *container* JSP provê uma lista de objetos instanciados, chamados de objetos implícitos. É através do objeto aplicação *(application object)* que são rastreadas as informações de um cliente específico entre múltiplas requisições.

CESPE - 2011 - CBM-DF

O uso de Javabeans, o controle de transferência entre as páginas e o suporte independente de *applets* Java pelos *browsers* são possibilidades proporcionadas pela *action tag* da JSP.

[15] FCC - 2009 - TRT-MG

NÃO possui uma habilidade de armazenar e recuperar valores de atributos arbitrários o objeto implícito de JSP

a) *session*

b) *request*

c) *exception*

d) *application*

e) *pageContext*

Expression Language

- ✓ interação entre apresentação e lógica da aplicação
- ✓ surgiu com a JSTL 1.0
  - inspirada em ECMAScript (Javascript) e Xpath EL
- ✓ adotada como parte da JSP 2.0 (JEE 1.4.2)
  - reduzir o uso de Scriptings JSP
  - compõe a API JSP
  - especificação própria no JEE 7 (EL 3.0 - JSR-341)
- ✓ acesso dinâmico e facilitado a JavaBeans
  - `${person.name }`
  - `<%= ((Person) request.getAttribute("person")).getName() %>`

▷ **Sintaxe**

- ✓ JSP 2.0 `${expr}`
- ✓ desativação
  - web.xml
    - ⇒ el-ignored de jsp-property-group
    - ⇒ scripting-invalid
  - diretiva page: isELIgnored

JSP 2.1 `${expr}` `#{expr}`

immediate deferred

Expression Language

- **Sintaxe**
  - ✓ acessando propriedades
    - ${param.name }
    - ${param['email']}
  - ✓ operador ponto
    - map, javabean, ou objeto implícito
  - ✓ operador colchetes
    - + List ou array
      - ${musicList[15]}
      - ${musicList['15']}
    - aninhamento
      - ${musicList[rockDisks[8]]}
      - ${musicList[rockDisks[8] + 4]}
  - ✓ operadores (Rvalue)

aritmético	+ - (binary) * / div % mod - (unary)
lógico	and && or || not !
comparação	== eq != ne < lt > gt <= ge >= le
condicional	A ? B : C
vazio	empty

- **Unified Expression Language (Java EE 5)**
  - ✓ EL JSP 2.1
    - EL JSP 2.0 + EL JSF 1.0
    - JSP: avaliação imediata ${expr}
      - ⇒ read-only: get()
      - ⇒ fase de tradução do JSP
    - JSF: avaliação adiada #{expr}
      - ⇒ get() e set()
      - ⇒ fases do JSF + fase execução do JSP

Expression Language

- **Unified Expression Language**
  - ✓ define dois tipos de expressão
    - Value Expressions
      - ✓ obtém (get) ou define (set) um valor
      - ✓ podem acessar JavaBeans, Enums, Collections, objetos implícitos
        - Rvalue ⇒ read-only immediate ${expr}
        - Lvalue ⇒ get() e set() deferred #{expr}
      - ✓ usada em texto estático ou atributos de tag (standard/custom)

```
<some:tag value="${expr}">
      some text ${expr} some text
      <another:tag value="${expr}text${expr}"/>
</some:tag>
```

    - Method Expressions
      - ✓ suporta métodos com avaliação adiada (deferred) #{expr}
        - invocados em fases pós-tradução
      - ✓ usado em atributos de tags JSF
        - tratamento de eventos
        - validação de componentes

EL: Objetos implícitos

- ✓ disponíveis para uso em expressões EL
- ✓ não são os mesmos do JSP
  - exceto pageContext
- ✓ objetos do tipo Map
  - conjunto de pares chave/valor

▷ objetos que mapeiam atributos de escopos

▸ pageScope	⇒ PageContext.getAttribute(String nome)
▸ requestScope	⇒ ServletRequest.getAttribute(String nome)
▸ sessionScope	⇒ HttpSession.getAttribute(String nome)
▸ applicationScope	⇒ ServletContext.getAttribute(String nome)

✓ não referenciam os objetos page, request, session e application

```
<user
      login="${applicationScope.person.user}"
      pass="${applicationScope.person.password}"/>
```

EL: Objetos implícitos

- objetos que mapeiam parâmetros e outras informações

param	⇒ ServletRequest.getParameter(String nome)
paramValues	⇒ ServletRequest.getParameterValues(String nome)
initParam	⇒ ServletContext.getInitParameter(String nome)
header	⇒ HttpServletRequest.getHeader(String nome)
headerValues	⇒ HttpServletRequest.getHeaders(String nome)
cookie	⇒ HttpServletRequest.getCookies()

```
Nome:  ${param.name}
Pedido nr: ${cookie.PedidoID.value}
item 1: ${paramValues.item[0]}
item 2: ${paramValues.item[1]}
Total: ${paramValues.valorItem[0] + paramValues.valorItem[1]}

Browser: ${header.user-agent}
webmaster: ${initParam.mainEmail}
```

EL: Objetos implícitos

- Objeto pageContext
  - ✓ mesmo objeto implícito JSP
    - instância de *javax.servlet.jsp.PageContext*
  - ✓ provê acesso a objetos implícitos JSP
    - application
    - response
    - page
    - session
    - config
    - pageContext
    - request
    - out
    - exception

```
<requestInfo>
    porta: ${pageContext.request.serverPort}
    protocolo: ${pageContext.request.protocol}
</requestInfo>
<sessionInfo>
    ID seção: ${pageContext.session.id}
    Tempo inatividade: ${pageContext.session.maxInactiveInterval} segundos
</sessionInfo>
<appInfo>
    Container: ${pageContext.servletContext.serverInfo }
    Versão Servlet suportada: ${pageContext.servletContext.majorVersion }
</appInfo>
```

[16]

CESPE - 2010 - TCU

O uso de expressões da *unified expression language* deve ser evitado dentro do código de classes Java, mas tais tipos de expressões são adequados e devem ser usados em páginas JSF, entre outras razões, por possibilitarem maior legibilidade ao código e constituírem alternativa mais simples ao uso de *tags* como <jsp:getProperty>.

CESPE - 2010 - MPU

Para que métodos estáticos de classes Java sejam executados a partir das funções da linguagem de expressão em JSP, é necessário que o nome da função coincida com o nome do método da classe Java.

[17]

CESPE - 2012 - TJ-AL [adaptada]

Em < h:inputText id="name" value= "#{customer.name}" /> o atributo value da tag h:inputText faz referência a uma expressão de avaliação tardia que aponta para a propriedade name do bean customer.

CESPE - 2012 - TJ-AL [adaptada]

As expressões que são avaliadas imediatamente pelo compilador podem atuar como expressões rvalue e lvalue.

JSP Standard Tag Library

- ✓ coleção de tags XML para tarefas comuns
  - iteração, controle de fluxo, manipulação de dados
  - reduzir o uso de Scriptings JSP
  - complementa a EL
- ✓ não é parte da API JSP (JSR 52)
  - mecanismo de extensão JSP
- ✓ Composição
  - tag handler (código Java)
  - tag library

▷ **Tag Libraries**

- ✓ coleção de actions relacionadas a áreas funcionais
- ✓ invocadas pela diretiva **taglib** (standard) ou **namespace** (XML)

Área	URI	Prefixo
▸ Core	http://java.sun.com/jsp/jstl/core	c
▸ Processamento XML	http://java.sun.com/jsp/jstl/xml	x
▸ Internacionalização I18N	http://java.sun.com/jsp/jstl/fmt	fmt
▸ SQL	http://java.sun.com/jsp/jstl/sql	sql
▸ Funções	http://java.sun.com/jsp/jstl/functions	fn

JSP Standard Tag Library

- **Core Tag Library**
  - ✓ ações de propósito geral

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html  xmlns:c="http://java.sun.com/jsp/jstl/core">
```

▸ variáveis	remove set
▸ condicional	if choose when otherwise
▸ iteração	forEach forTokens

▸ outros	catch out
▸ URL	import redirect url param

```
<c:set var="bookId" scope="page" value="#{BooksBean.bookID}"/>
<c:out value="${param.title }"  escapeXml="true" />
```

```
<c:forEach var="phone" items="${address.phones}">
    ${phone}"<br/>
</c:forEach>
<c:forEach var="x" begin="1" end="50" step="5">
    <c:out value="${x}"/>,
</c:forEach>
```

```
<c:if test="${param.user=='ken' && param.password=='blackcomb'}">
    You logged in successfully.
</c:if>
<c:choose>
    <c:when test="${count == 0}" >
        No records matched your selection.
    </c:when>
    <c:otherwise>
        ${count} records matched your selection.
    </c:otherwise>
</c:choose>
```

```
<c:import url="http://www.safaribooks.com/library/bookList.jsp" var="list" />
<c:url var="url" value="/catalog" >
    <c:param name="Add" value="${bookId}" />
</c:url>
```

JSP Standard Tag Library

- **XML Tag Library**
  - ✓ manipula elementos XML
  - ✓ ações baseadas no Xpath para acessar elementos XML
    - similar a tag library Core
  - ✓ transformação usando documentos XSLT

```
<%@taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html  xmlns:x="http://java.sun.com/jsp/jstl/xml">
```

▸ core	out parse set
▸ transformação	transform param

▸ fluxo de controle	if choose when otherwise forEach

```
<c:import url="books.xml" var="bookUrl"/>
<x:parse doc="${bookUrl}" var="doc"/>

<x:forEach var="b" select="$doc/books/book">
    <x:if select="$doc/books/book/price > 10">
        <tr>
            <td><x:out select="$b/@isbn"/></td>
            <td><x:out select="$b/title"/></td>
            <td><x:out select="$b/@price"/></td>
        </tr>
    </x:if>
</x:forEach>
```

```
<c:import url="invoice.xml" var="xmlDoc" />
<c:import url="templateHtml.xsl" var="xslDoc" />

<x:transform doc="${xmlDoc}" xslt="${xslDoc}" >
    <x:param name="client" value="${clientName }"/>
    <x:param name="invoice" value="${invoiceID }"/>
</x:transform>
```

JSP Standard Tag Library

- **I18N Tag Library**
  - ✓ formatação
    - data, números, moedas, etc
  - ✓ internacionalização
    - timezone, locales, messaging

```
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<html  xmlns:fmt="http://java.sun.com/jsp/jstl/fmt">
```

▸ locale	setLocale requestEncoding
▸ messaging	bundle message param setBundle

▸ formatação	formatNumber formatDate parseDate parseNumber setTimeZone timeZone

```
<c:set var="now" value="${new java.util.Date()}" />

<fmt:formatDate type="time" timeStyle="short"
                value="${now}" />
<fmt:formatDate type="date" dateStyle="short"
                value="${now}" />

<fmt:setTimeZone value="GMT-8" />
<fmt:formatDate type="both" dateStyle="short"
                timeStyle="short"  value="${now}" />
```

```
<fmt:setLocale value="en_us" />
<fmt:formatNumber value="20.50" type="currency" />
<fmt:setLocale value="en_gb" />
<fmt:formatNumber value="20.50" type="currency" />
```

```
<fmt:setLocale value="hi_IN" />
<fmt:setBundle basename="ResourceBundles.ErrorMessages"
               var="errorMessages" />
<fmt:message key="ER4582" bundle="${errorMessages}" />
```

JSP Standard Tag Library

- **SQL Tag Library**
  - ✓ acesso a bancos de dados
    - funcionalidades básicas para interação com SGBD-R
    - protótipos e aplicações simples
    - produção: EJBs, JPA, etc

```
<%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
<html  xmlns:sql="http://java.sun.com/jsp/jstl/sql">
```

- datasource setDataSource

- SQL query
  update
    dateParam
    param
  transaction

```
<sql:setDataSource var="snapshot" driver="com.mysql.jdbc.Driver"
        url="jdbc:mysql://localhost/TEST"
        user="root"  password="pass123"/>
```

```
<sql:transaction dataSource="${snapshot}">
    <sql:update var="qEmp">
        UPDATE Employees  SET empAddress = ?  WHERE empID = ?
            <sql:param value="${empAddress}" />
            <sql:param value="${empId}" />
    </sql:update>
</sql:transaction>
```

```
<sql:query dataSource="${snapshot}" var="result">
    SELECT * from Employees where firedDate = ?
        <sql:dateParam value="${new java.util.Date()}"
                type="DATE" />
</sql:query>
```

JSP Standard Tag Library

- **Function Tag Library**
  - ✓ não são tags
    - usada com EL
  - ✓ manipulação de Strings

```
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<html xmlns:fn="http://java.sun.com/jsp/jstl/functions">
```

contains	toLowerCase	trim	endsWith
containsIgnoreCase	substring	replace	split
length	substringAfter	indexOf	join
toUpperCase	substringBefore	startsWith	escapeXml

```
<c:set var="nameForm" value="${fn:toUpperCase(param.name)}"/>
<c:set var="name" value="${applicationScope.contato.name}"/>

<c:if test="${fn:contains(name, 'nameForm')}">
    <p>Nome correto<p>
</c:if>

<c:import url="/contato.txt" var="contatoTXT" />
<c:set var="contato" value="${fn:split(contatoTXT, '; ')}" />
Nome: ${contato[0]}
Local: ${contato[1] }

<c:set var="string1" value="This is first String."/>
<c:set var="string2" value="${fn:replace(string1, 'first', 'second')}" />

Qtde atletas:  ${fn:length(athletes)} - País: ${country}
```

JSP Custom Tags

- ✓ mecanismo de extensão JSP
- ✓ bibliotecas com tags
  - terceiros ou próprias
- ✓ Composição
  - tag handler (classe Java)
  - tag library descriptor (TLD)
  - diretiva taglib na página JSP

▷ **Tag Handler**

- ✓ classe java
  - pacote *javax.servlet.jsp.tagext*
  - implementa a interface SimpleTag (JSP 2.0)
    - ⇒ método doTag
    - ⇒ métodos setter para atributos (JavaBean)
    - ⇒ classe SimpleTagSupport
  - localização
    - ⇒ WEB-INF/classes
    - ⇒ WEB-INF/lib (JAR)

JSP Custom Tags

- **Tag Library Descriptor (TLD)**
  - ✓ documento XML com extensão .tld
    - localização
      - ⇒ WEB-INF
      - ⇒ META-INF do JAR (WEB-INF/lib)
  - ✓ mapeamento web-xml (automático JSP 2.0)
  - ✓ elementos `<taglib>` `<uri>` `<function>` `<tag>`
  - ✓ <tag>
    - elemento mais importante na TLD
    - sub-elementos

      `<name>`

      `<tag-class>`

      `<body-content>`

      `<attribute>`

JSP Custom Tags

```
public class HelloTag extends SimpleTagSupport {
  private String message;
  public void setMessage(String msg) {
    this.message = msg;
  }
  public void doTag() throws JspException, IOException {
    StringWriter sw = new StringWriter();
    if (message != null) {
      JspWriter out = getJspContext().getOut();
      out.println( message );
    }
    else {
      getJspBody().invoke(sw);
      getJspContext().getOut().println(sw.toString());
    }
  }
}
```

```
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_1.xsd"
        version="2.1">
    <uri>myTagLibrary</uri>
    <tag>
        <name>Hello</name>
        <tag-class>handlersClass.HelloTag</tag-class>
        <body-content>scriptless</body-content>
        <attribute>
            <name>message</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

```
<%@ taglib prefix="ex" uri="myTagLibrary"%>
<html>
<head>
        <title>A sample custom tag</title>
</head>
<body>
        <ex:Hello message="This is custom tag" />
</body>
</html>
```

[18] FCC – 2010 – TRT-8ª

A biblioteca de tags padrão do Java Server Pages (JSTL) é uma coleção de tags padronizadas para tarefas comuns a muitas aplicações JSP. Estas tags estão divididas em 5 áreas funcionais:

a) Web, XML, Banco de Dados, Internacionalização (I18n) e Funções.

b) Core, XML, Banco de Dados, Internacionalização (l18n) e Funções.

c) Core, Estatística, Banco de Dados, Internacionalização (I18n) e Funções.

d) Web, Matemática, Banco de Dados, Internacionalização (I18n) e Funções.

e) Core, Web, Matemática, Internacionalização (I18n) e Funções.

[19] CESPE - 2009 - CEHAP-PB

Assinale a opção incorreta acerca de JSTL.

a) O JSTL não possui suporte nativo a SQL. Para tanto, é utilizada em conjunto com a biblioteca nativa SQL do J2EE.

b) Uma página JSTL pode ser definida como uma página JSP contendo um conjunto de tags JSTLs.

c) Em uma página JSTL, cada tag realiza determinado tipo de processamento.

d) O JSTL permite ao programador escrever páginas JSPs sem necessariamente utilizar códigos Java, facilitando a integração entre web designers e programadores.

[20] CESPE - 2008 - MPE-RR

Na linha 13 do trecho de código mostrado, é utilizada uma *tag* de uma biblioteca de *tags* padronizada JSLT.

```
1   <%@ page contentType="text/html; charset=UTF-8" %>
2   <%@ taglib prefix="s" uri="/struts-tags" %>
3   <html>
4   <head>
5   <title>Sign On</title>
6   </head>
7
8   <body>
9         <s:form action="Login">
10              <s:textfield key="username" />
11              <s:password key="password" />
12              <s:submit />
13        </s:form>
14  </body>
15  </html>
```

[21] CESPE – 2010 - MPU

As *tags* personalizadas são produzidas em arquivos TLD *(tag library description)*. O código a seguir é um exemplo de *tag* personalizada no JSF 1.0.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
PUBLIC "-//sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
"http://java.sun.com/dtd/seb-jsptaglibrary_1_2.dtd">
<taglib>
      <tlib-version>1.0</tlib-version>
      <jsp-version>1.2</jsp-version>
      <short-name>teste</short-name>
      <uri>http://exemplojsf.com.br/teste</uri>
      <tag>
            <name>testeProva</name>
            <tag-class>br.com.exemplojsf.TesteProvaTag</tag-class>
            <attribute>
                  <name>separator</name>
            </attribute>
      </tag>
</taglib>
```

[22]

CESPE - 2012 - PERITO CRIMINAL-CE

A JSP permite introduzir tags customizadas à sua biblioteca e, assim, estender facilidades à linguagem. Entre outros benefícios dessa prática estão a eliminação de scriptles em aplicações JSP, reúso e sintaxe similar à do HTML.

CESPE - 2007 - CBM-DF

Uma página JSP pode conter tags que, quando encontradas, causam a execução de código Java. Essas tags possibilitam a inserção de lógica dentro das páginas e podem ser agrupadas em bibliotecas. As bibliotecas sendo usadas podem ser identificadas via tags no arquivo web.xml.

CESPE - 2011 - Correios

Ao se usar tag personalizada **JSP,** é suficiente carregar uma **URL** que indique a localização do arquivo **TLD** para a biblioteca que se deseja acessar.

JavaServer Faces

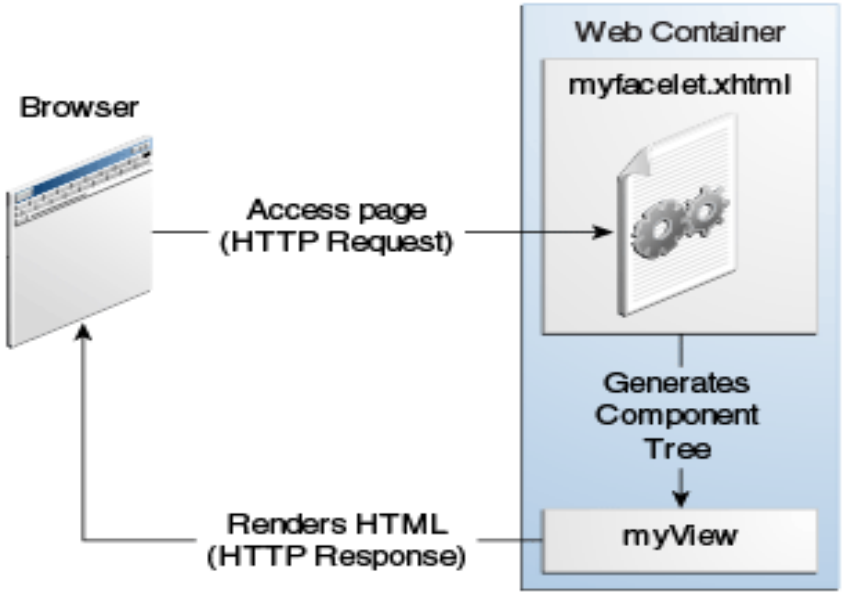
- rodam no lado servidor
  - ✓ Componente web que roda no container web
- Definição

  é um framework baseado em componentes para interface de usuário utilizado para construir aplicações web.
- Composição
  - ✓ API
    - ⇒ representar componentes visuais e gerenciar estados
    - ⇒ tratar eventos
    - ⇒ validação e conversão de dados
  - ✓ bibliotecas de tags
    - ⇒ componentes UI para páginas web e conexão com objetos server-side
- Especificação JSF
  - ✓ **JEE 5**: JavaServer Faces 1.2 (JSR 252)
  - ✓ **JEE 6**: JavaServer Faces 2.0 (JSR 314)
  - ✓ **JEE 7**: JavaServer Faces 2.2 (JSR 344)

JavaServer Faces

- implementação
  - ✓ Mojarra (implementação de referência)
  - ✓ Apache MyFaces
    - ⇒ Geronimo e TomEE
  - ✓ componentes sofisticados
    - ⇒ RichFaces, PrimeFaces, IceFaces, etc

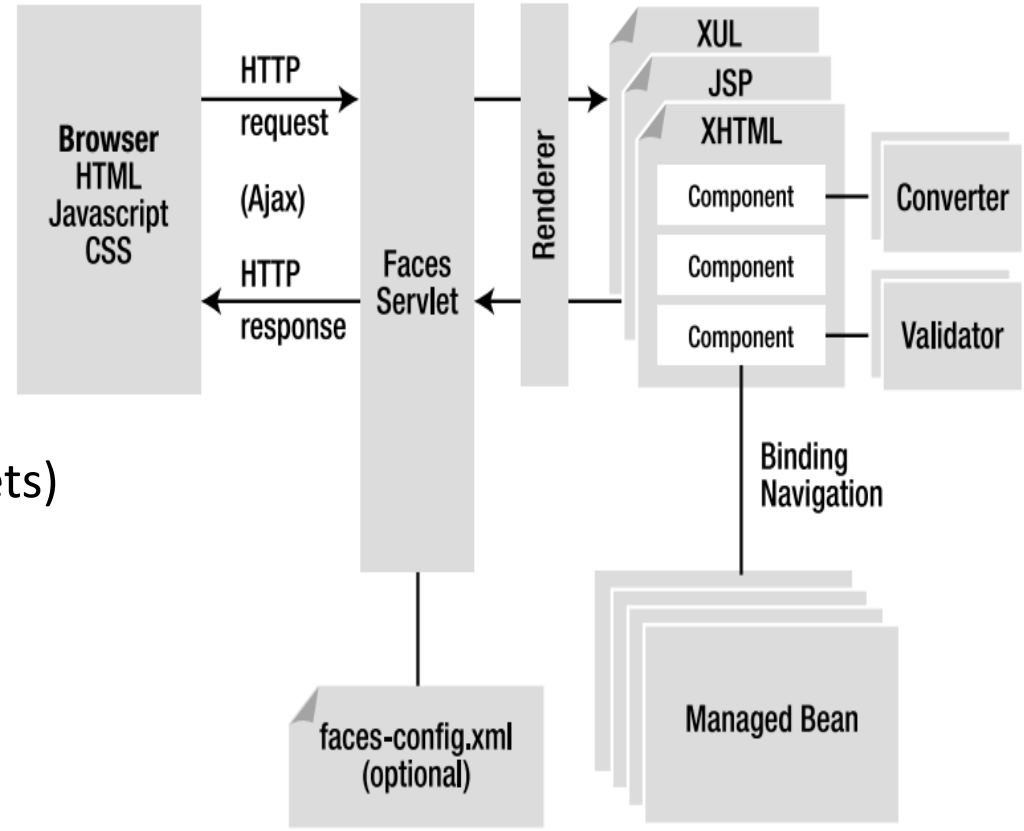
- Aplicação JSF
  - ✓ web page
    - ⇒ componentes UI
    - ⇒ managed beans
  - ✓ XML para configurações
    - ⇒ web.xml
    - ⇒ faces-config.xml (opcional)



- Benefícios (tutorial)
  - ✓ clara separação entre comportamento (lógica) e apresentação

JavaServer Faces: Arquitetura

- ▷ Faces Servlet
  - ✓ controla o ciclo de vida JSF
    - ⇒ intercepta requisições HTTP
  - ✓ gerado pelo container
- ▷ User Interface
  - ✓ páginas web e componentes GUI
  - ✓ PDL ou VDL
    - ⇒ JEE 5 (JSF 1.2): JSP
    - ⇒ JEE 6 (JSF 2.0): XHTML (facelets)
- ▷ Tratadores
  - ✓ renders, converters e validators
- ▷ Managed beans
  - ✓ lógica da aplicação (negócio)
  - ✓ navegação entre páginas



[23] FCC - 2011 - TRT - 14ª

Em relação a **frameworks Java**, considere:

I. Associa os eventos do lado cliente com os manipuladores dos eventos do lado do servidor.

II. Fornece separação de funções que envolvem a construção de aplicações *Web*.

III. Inclui um conjunto padrão de componentes de interface de usuário que possibilitam validação padronizada.

Os itens I, II e III, referem-se a

a) **EJB**, apenas.
b) **JSF**, apenas.
c) *Hibernate*, apenas.
d) **EJB** e **JSF**.
e) **JSF** e *Hibernate*.

[24]

FCC - 2010 - TRT-PI [adaptada]

JavaServer Faces é um *framework* MVC utilizado no desenvolvimento de aplicações para a internet de forma visual, que utiliza o recurso de arrastar e soltar os componentes na tela para definir suas propriedades.

CESPE - 2010 - TRE-BA

Uma página JSF pode ser composta de conteúdos estáticos, como *tags* HTML e códigos dinâmicos.

[25]

CESPE - 2009 - SECONT-ES

O JSF é um framework web embasado em interface gráfica, capaz de renderizar componentes e manipular eventos em aplicações web no padrão Java EE, no qual os componentes JSF são orientados a eventos. O JSF fornece, ainda, mecanismos para conversão, validação, execução de lógica de negócios e controle de navegação.

FUMARC - 2014 - PBH [adaptada]

JSF é um *framework* para desenvolvimento de aplicações Web em Java, baseado no modelo MVC, para construção de interfaces com os usuários por meio de componentes visuais.

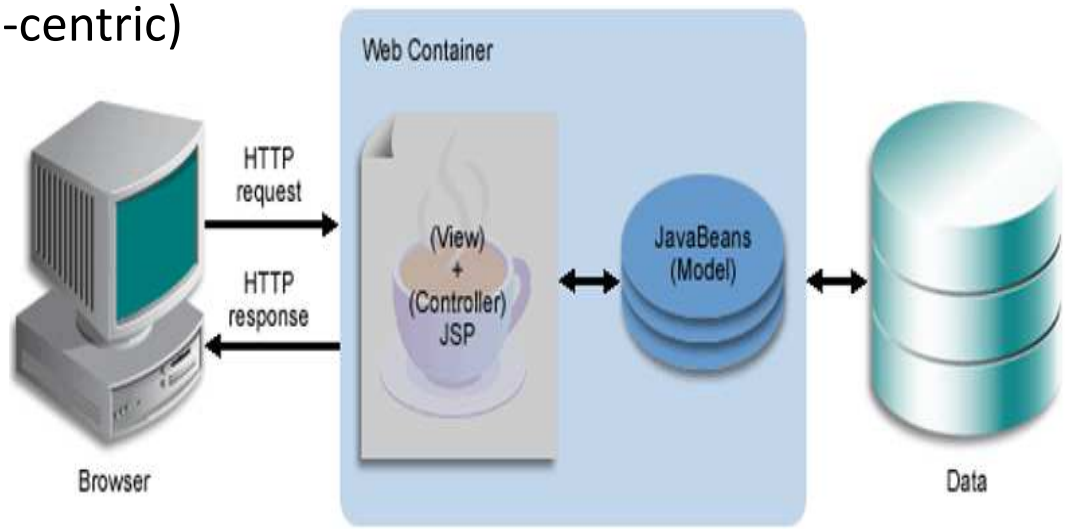
Model-View-Controller

- ✓ padrão arquitetural
- ✓ divide a aplicação em 3 funções ou componentes
  - ⇒ modelo, visão e controlador
- ✓ separar o modelo da apresentação
  - ⇒ UI: visão + controlador (Fowler)
  - ⇒ baixo acoplamento
- ✓ PoEAA e POSA

▷ MODEL 1

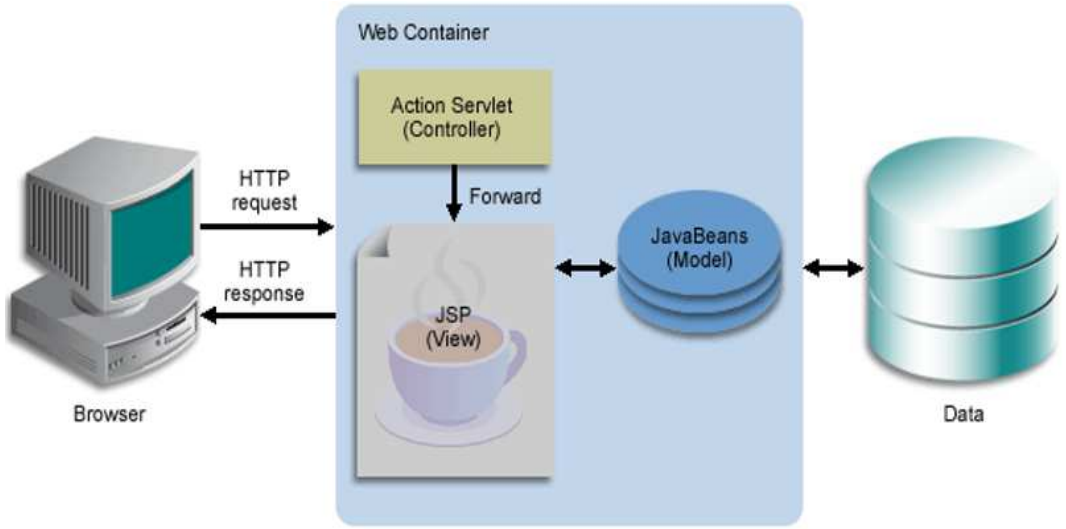
não considerado como MVC (wikipedia)

- ✓ não há separação clara entre visão e controlador
- ✓ modelo centrado no JSP (page-centric)
  - ⇒ funções de view/controller
  - ⇒ elementos HTML (output)
  - ⇒ trata requisições HTTP
  - ⇒ invoca lógica de negócio
- ✓ pequenas aplicações
  - ⇒ escalabilidade ruim
  - ⇒ baixa manutenibilidade

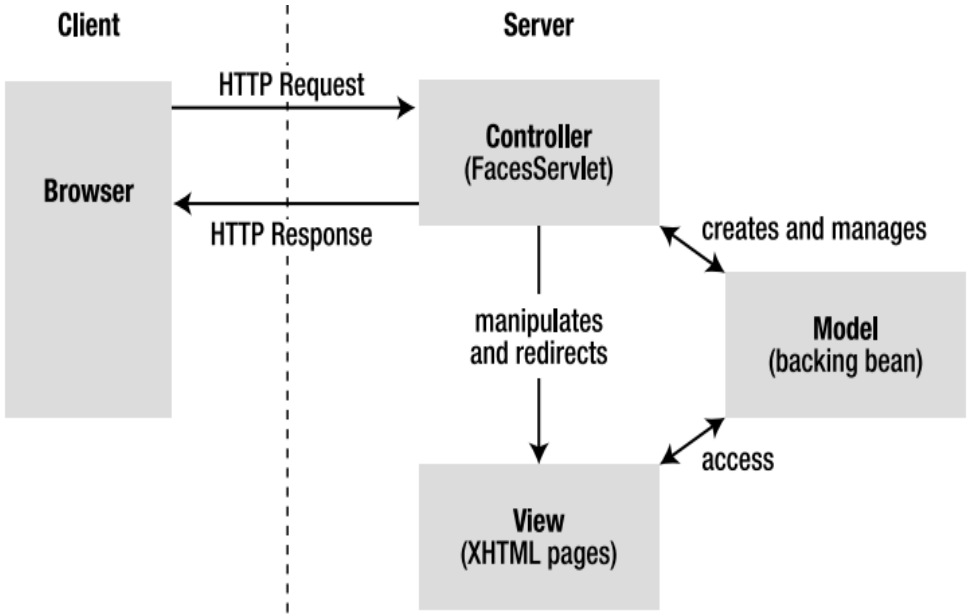


Model-View-Controller

- **MODEL 2**
  - ✓ separa a view do controller
    - ⇒ model: JavaBeans/EJB
    - ⇒ view: JSP/XHTML
    - ⇒ controller: Servlet
  - ✓ Servlet > Front Controller
    - ⇒ ponto único de entrada
  - ✓ Servlet-centric



- **Java-Server Face**
  - ✓ baseada no Model 2 (MVC2)
  - ✓ MVC orientado a componentes UI
    - ⇒ **controller**: FacesServlet
      - ▹ JSF lifecycle, JSF Framework
    - ⇒ **view**: JSP/Facelets(XHTML)
      - ▹ UI componentes tree
    - ⇒ **model**: ManagedBeans
      - ▹ backing bean
      - ▹ EJB/JPA



[26] FCC - 2012 - TJ-PE

No JSF, o componente *Controller do MVC* é composto por uma classe servlet, por arquivos de configuração e por um conjunto de manipuladores de ações e observadores de eventos. Essa *servlet* é chamada de

a) ControllerServlet.

b) Facelet.

c) HttpServlet.

d) FacesConfig.

e) FacesServlet.

[27] FCC - 2013 - TRT-PR

Uma aplicação utilizando o *framework* JSF e a IDE NetBeans gera automaticamente dois componentes essenciais assim descritos:

I. É responsável por receber requisições dos componentes *View* do MVC, redirecioná-las para os *beans* gerenciados (*managed beans*) do componente *Model* do MVC e responder a essas requisições.

II. É o arquivo principal de configuração de uma aplicação *web* que utiliza o *framework* JSF. É responsável por descrever os elementos e sub-elementos que compõem o projeto, tais como as regras de navegação, *beans* gerenciados, configurações de localização etc.

As descrições I e II referem-se, respectivamente, aos componentes

a) servlet Controller.java e ao arquivo faces_config.xml

b) FaceletServlet e ao arquivo web_config.xml.

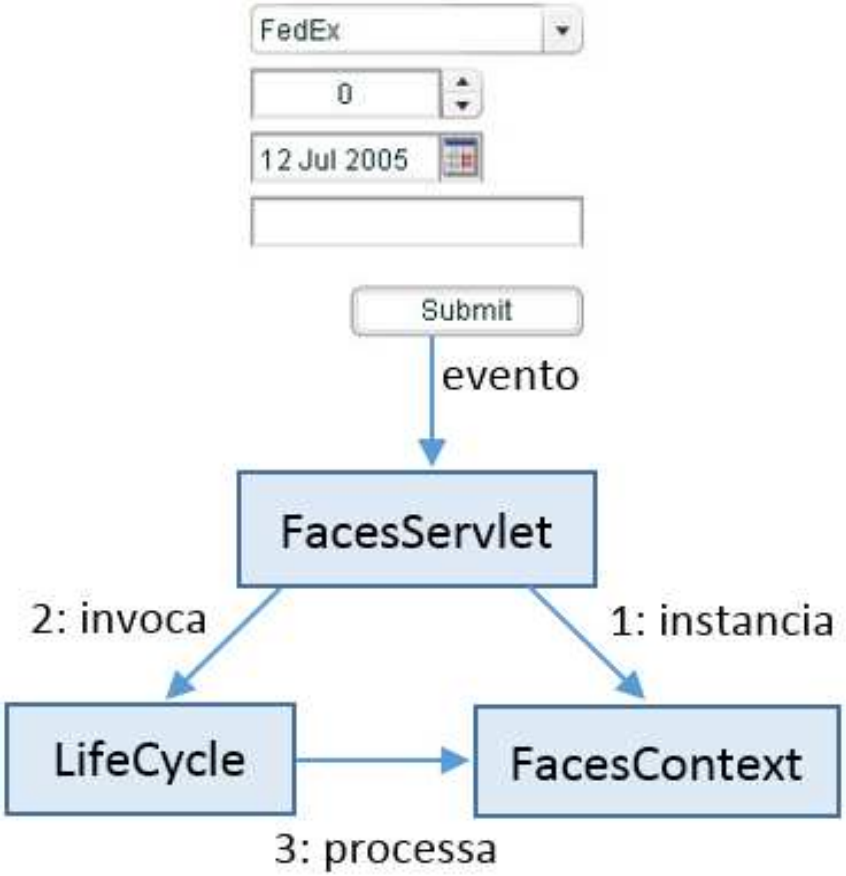
c) FacesServlet e ao arquivo faces-config.xml.

d) servlet Controller e ao arquivo web-config.xml.

e) servlet Facelet e ao arquivo web.xml.

JavaServer Faces: FacesServlet

- ✓ engine (servlet) da aplicação JSF (controller)
  - ⇒ javax.faces.webapp.FacesServlet
  - ⇒ implementa javax.servlet.Servlet
    - ▹ *init(ServletConfig), destroy()* e *service(ServletRequest, ServletResponse)*
- ✓ examina requisições HTTP
  - ⇒ mapear no web.xml
- ✓ interno ao framework
  - ⇒ gerado pelo container
    - ▹ *javax.faces.context.FacesContext*
  - ⇒ configurações via metadados
    - ▹ faces-config.xml (JSF 1.2)
    - ▹ annotations ou faces-config.xml (JSF 2.0)
- ✓ gerencia o ciclo de vida JSF
  - ✓ controla/invoca objetos do ciclo de vida
  - ✓ **ciclo de vida JSF != Servlet**
    - ⇒ servlets = métodos
    - ⇒ JSF = objetos



JavaServer Faces: FacesServlet

- **web.xml**
  - ✓ obrigatório em JSF 2.0 (JEE6)
  - ✓ mapear como servlet
    - ⇒ <servlet-name> e <servlet-mapping>

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>
    javax.faces.webapp.FacesServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

- **faces-config.xml (opcional)**
  - ⇒ JEE 5 (JSF 1.2): obrigatório
  - ✓ define configurações da aplicação
    - ⇒ MBeans, validators, converters, etc
    - ⇒ navegação, localização, etc
  - ✓ pode ser fragmentado
    - ✓ arquivos separados por grupos de configurações
    - ✓ parâmetro javax.faces.CONFIG_FILES (web.xml)
      - ⇒ lista de arquivos de configuração

```
<context-param>
  <param-name>
    javax.faces.CONFIG_FILES
  </param-name>
  <param-value>
    WEB-INF/main-faces-config.xml,
    WEB-INF/navigation-faces-config.xml,
    WEB-INF/managed-beans-faces-config.xml,
    WEB-INF/converters-faces-config.xml,
    WEB-INF/validators-faces-config.xml
  </param-value>
</context-param>
```

JavaServer Faces: faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.2" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd">

    <application>
        <el-resolver>jsf.util.JsfCrudELResolver</el-resolver>
        <locale-config>
            <default-locale>en</default-locale>
            <supported-locale>en</supported-locale>
            <supported-locale>es</supported-locale>
            <supported-locale>pt-br</supported-locale>
        </locale-config>
    </application>

    <navigation-rule>
        <from-view-id>/logon.jsp</from-view-id>
        <navigation-case>
            <from-action>#{LogonForm.logon}</from-action>
            <from-outcome>success</from-outcome>
            <to-view-id>/storefront.jsp</to-view-id>
        </navigation-case>
        <navigation-case>
            <from-action>#{LogonForm.logon}</from-action>
            <from-outcome>failure</from-outcome>
            <to-view-id>/logon.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

    <validator>
        <validator-id>FormatValidator</validator-id>
        <validator-class>validators.FormatValidator</validator-class>
        <attribute>
            <attribute-name>formatPatterns</attribute-name>
            <attribute-class>java.lang.String</attribute-class>
        </attribute>
    </validator>

    <converter>
        <converter-id>CreditCardConverter</converter-id>
        <converter-class>converters.CreditCardConverter</converter-class>
    </converter>

    <managed-bean>
        <managed-bean-name>personBean</managed-bean-name>
        <managed-bean-class>jsfks.PersonBean</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
        <managed-property>
            <property-name>personName</property-name>
            <property-class>java.lang.String</property-class>
            <value>JavaJoe</value>
        </managed-property>
    </managed-bean>

</faces-config>
```

[28]

CESPE - 2013 - STF

A configuração do controlador do JSF é realizada no *servlet* contido no arquivo *web.xml*. Essa servlet é responsável por receber as requisições e delegá-las ao núcleo do JSF.

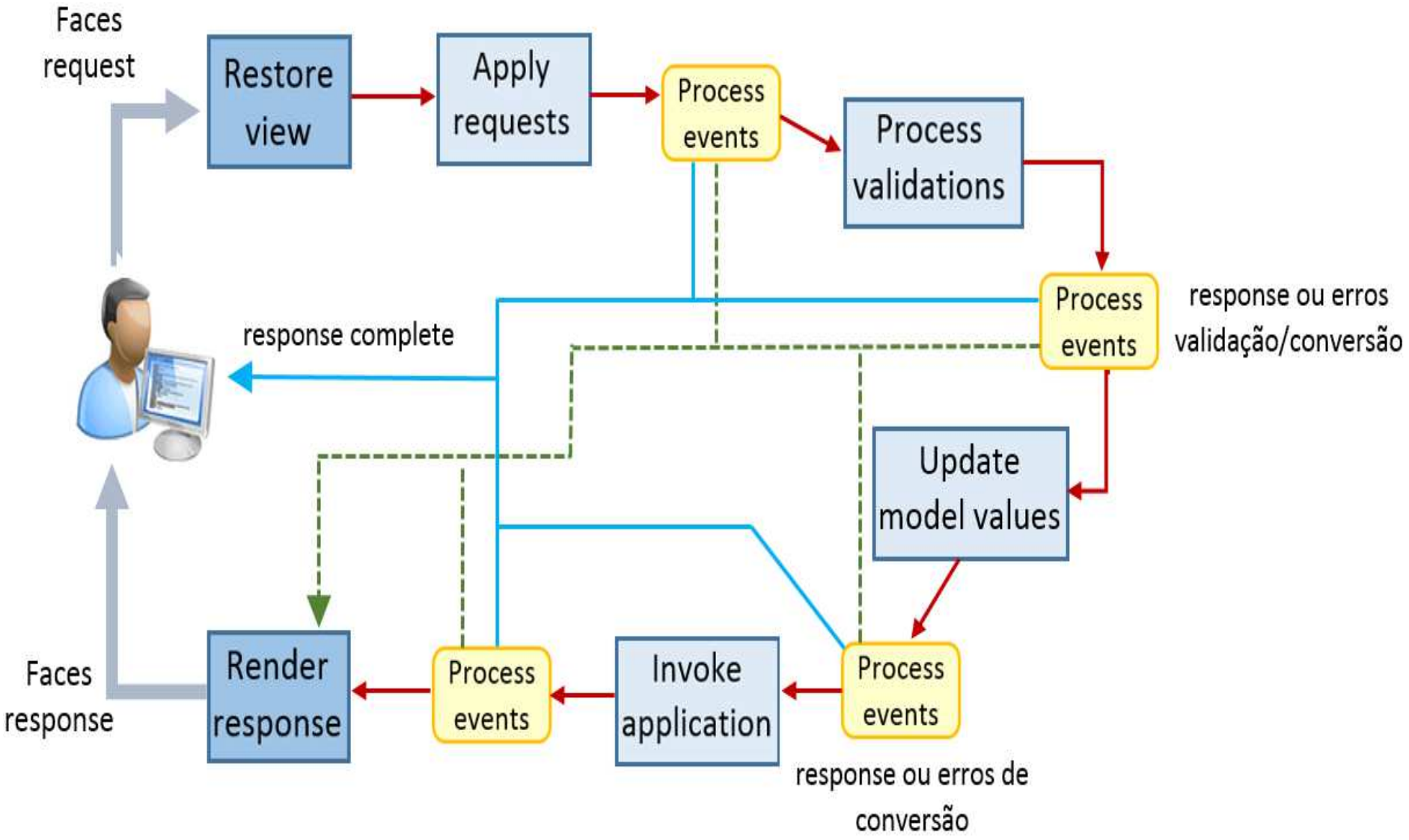
CESPE - 2014 - TJ-SE

Antes de uma aplicação *web* desenvolvida nos moldes da JSF executar sua primeira página *web*, uma instância *FacesServlet* é executada, a fim de gerenciar as requisições dessa aplicação.

JavaServer Faces: Ciclo de vida

- ✓ consiste de seis fases
  1. restore view
  2. apply request values
  3. process validation
  4. update model values
  5. invoke application
  6. render response
- ✓ ciclo de vida de uma aplicação JSF
  - ⇒ início: HTTP request do cliente
    - ▹ initial request: restore view e render response
    - ▹ postback request: todas as fases
  - ⇒ fim: HTTP response do servidor
- ✓ objeto Lifecyle
  - ⇒ classe abstrata
    - ▹ *`javax.faces.lifecycle.Lifecycle`*
  - ⇒ executa as fases do ciclo de vida
    - ▹ instanciado via factory para cada fase
    - ▹ principais métodos
      - **execute**(FacesContext): 5 primeiras fases
      - **render**(FacesContext): Render response

JavaServer Faces: Request LifeCycle



JavaServer Faces: Request LifeCycle

▷ 1. Restore View

- ✓ constrói/recupera a árvore de componentes (viewID)
  - ⇒ componentes de comando geram eventos de ação
- ✓ liga tratadores de eventos e validadores a componentes
- ✓ UIviewRoot armazenada em FacesContext

▷ 2. Apply Request Values

- ✓ extrai os valores de request parameter
  - ⇒ cookies ou headers
- ✓ método processDecodes()
  - ⇒ conversão para o tipo do componente JSF
    - ▹ se falha: erro é armazenado em FacesContext
- ✓ atualiza estado de componentes UI
- ✓ verifica o atributo immediate
  - ⇒ true: validação/conversão ocorre nesta fase
    - ▹ botão Cancela

JavaServer Faces: Request LifeCycle

▷ 3. Process Validation

- ✓ executa os validadores registrados
- ✓ verifica regras/restrições de valores
  - ⇒ pré-definidas ou definidas pelo desenvolvedor
- ✓ método processValidators()
  - ⇒ marca o atributo valid dos componentes
  - ⇒ se falha: armazena erro em FacesContext
    - ▹ chama a fase RenderResponse

```
<h:inputText id="helloInput"
    value="#{helloBean.numControls}"
    required="true">
    <f:validateLongRange minimum="1"
        maximum="500"/>
</h:inputText>
```

▷ 4. Update Model Values

- ✓ atualiza as propriedades de objetos do modelo
  - ⇒ backing beans (bean properties)
- ✓ sintaxe e semântica da request são válidas
- ✓ método processUpdates()
  - ⇒ conversão para o tipo do atributo (propertie)
    - ▹ se falha: erro é armazenado em FacesContext
    - ▹ invoca a fase RenderResponse

JavaServer Faces: Request LifeCycle

▷ 5. Invoke Application

- ✓ lógica da aplicação é executada
  - ⇒ navegação, cálculos, etc
  - ⇒ submit de forms (botões/links)
    - ▹ atributo action
- ✓ trata eventos em nível de aplicação
  - ⇒ carrega os listeners
    - ▹ method binding
    - ▹ actionListener

```
<h:commandButton id="submitButton"
   value="Submit" action="#{userData.showResult}"
   actionListener="#{userData.updateData}" />
</h:commandButton>
```

```
<h:commandButton id="submitButton1"
   value="Submit" action="#{userData.showResult}" >
   <f:actionListener
      type="com.tutorialspoint.test.UserActionListener"/>
</h:commandButton>
```

▷ 6. Render Response

- ✓ produz uma view para responder ao cliente
  - ⇒ JSP/Facelet (XHTML)
- ✓ salva o estado dos componentes UI
- ✓ tratamento da request
  - ⇒ initial: renderiza componentes da árvore
  - ⇒ postback: salva estado, renderiza e exibe erros (h:message ou h:messages)

[29] FCC - 2011 – TER-AP

Considere:

O JSF extrai todos os valores digitados pelo usuário e guarda esse valor nos seus respectivos componentes. Se o valor digitado não coincidir com o componente, um erro vai ser adicionado na classe FacesContext *e será mostrado na fase* Render Response Phase*.*

No ciclo de vida do JSF trata-se de um evento típico da fase

a) Process Validations Phase.

b) Restore View Phase.

c) Apply Request Values Phase.

d) Update Model Values Phase.

e) Invoke Application Phase.

[30] FCC - 2012 - TRE-CE

No ciclo de vida do *Java Server Faces* trata-se da fase na qual o componente deve primeiro ser criado ou recuperado a partir do *FacesContext*, seguido por seus valores, que são geralmente recuperados dos parâmetros de *request* e, eventualmente, dos cabeçalhos ou *cookies* gerados. Trata-se da fase

a) Restore View.

b) Apply Request Values.

c) Process Validation.

d) Update Model Values.

e) Invoke Application.

[31] IADES - 2014 - TRE-PA

O clico de vida do JavaServer Faces (JSF) é composto por seis fases. Acerca desse tema, assinale a alternativa correta.

a) A fase em que se atualizam valores do modelo só é alcançada após todos os componentes JSF serem considerados válidos na fase que processa as validações.

b) Na fase em que se aplicam os valores de requisição, cada componente extrai seu novo valor e armazena em si próprio; porém, caso esse valor não seja do tipo String, uma mensagem de erro é gerada e associada ao componente.

[31] IADES - 2014 - TRE-PA

c) Quando é feita uma requisição a uma página JSF, sua implementação inicia a fase que processa as validações dos componentes.

d) Durante a fase de atualizar os valores do modelo, a implementação do JSF manipula eventos da aplicação, como ir para a próxima página por meio de um link.

e) Caso sejam encontrados erros nas fases em que as validações são processadas ou em que os valores do modelo são atualizados, tais erros são exibidos na página após a fase de renderizar a resposta, independentemente de a página conter um ou mais componentes <message/> ou <messages/>.

[32] CESPE - 2012 - ANAC

Com relação ao desenvolvimento Java, julgue os itens a seguir.

Na fase de submissão de valores via request do JSF, caso a conversão de um valor falhe, uma mensagem de erro associado com o componente é gerada, devolvida para FacesContext e exibida para o usuário, parando-se imediatamente o processamento a partir desse ponto.

JavaServer Faces: Managed Beans

- ✓ componentes JavaBeans
  - ⇒ gerenciados pelo container
    - ⇒ container IoC
      - ⇒ JSF Managed Bean Facility (MBF)
  - ⇒ registrados via XML ou annotations
  - ⇒ características
    - ▹ construtor no-args
    - ▹ conjunto de properties
      - ▹ atributo private + métodos get() e set() públicos
      - ▹ salva o estado de componentes da página web
      - ▹ dois tipos de associação
        - ▹ valor do campo HTML (value)
        - ▹ instância do componente UI (binding)
    - ▹ métodos com funções para os componentes
      - ▹ validação de dados, tratamento de eventos, navegação
- ✓ implementam lógica de negócios (model X view)
  - ⇒ separa a lógica da apresentação

JavaServer Faces: Managed Beans

▷ Backing Bean (JEE 5)

⇒ tutorial JEE5: MBean associado a componente UI

⇒ tutorial JEE6 e JEE7: não é explícito

⇒ conjunto de properties

✓ ligação da property com componente UI

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html">

<h:head>
      <title>Cadastro de Cliente</title>
</h:head>
<h:body>
      <h:form>
            Nome: <h:inputText id="nome"
                  binding="#{clienteBean.nome}" /><br />
            CPF: <h:inputText id="cpf"
                  binding="#{clienteBean.cpf}" /><br />
      </h:form>
</h:body>
</html>
```

```
import javax.faces.component.UIInput;

public class ClienteBean {

      private UIInput nome;
      private UIInput cpf;

      public UIInput getNome() {
            return nome;
      }
      public void setNome(UIInput nome) {
            this.nome = nome;
      }
      public UIInput  getCpf() {
            return cpf;
      }
      public void setIdade(UIInput cpf) {
            this.cpf = cpf;
      }
}
```

JavaServer Faces: Managed Beans

▷ XML (faces-config)

- ✓ elemento XML principal <managed-bean>
  - ⇒ representa uma instância de uma classe JavaBean
  - ⇒ Managed Bean Facility gerencia o ciclo de vida (IoC)

```
import javax.faces.component.UIInput;

public class ClienteBean {

      private UIInput nome;
      private UIInput cpf;

      public UIInput getNome() {
            return nome;
      }
      public void setNome(UIInput nome) {
            this.nome = nome;
      }
      public UIInput  getCpf() {
            return cpf;
      }
      public void setIdade(UIInput cpf) {
            this.cpf = cpf;
      }
}
```

```
<managed-bean eager="true">
  <managed-bean-name>clienteBean</managed-bean-name>
  <managed-bean-class>beanPackage.ClienteBean</managed-bean-class>
  <managed-bean-scope>application</managed-bean-scope>
  <managed-property>
    <property-name>nome</property-name>
    <value>José Augusto</value>
  </managed-property>
  <managed-property>
    <property-name>cpf</property-name>
    <value>12345678900</value>
  </managed-property>
</managed-bean>
```

```
<h:form>
   Nome: <h:inputText id="nome"
      value="#{clienteBean.nome}"/>
   CPF: <h:inputText id="cpf"
      value="#{clienteBean.cpf}"/>
</h:form>
```

Tutorial

JEE5: none, request, session, ou application

JEE6/7: none, request, **view**, session, ou application

JavaServer Faces: Managed Beans

▷ Annotations

- ✓ @ManagedBean
  - ⇒ pacote javax.faces.bean
  - ⇒ atributos: eager e name
- ✓ escopos
  - ⇒ tutorial JEE6: @ApplicationScoped, @SessionScoped, @ViewScoped, @RequestScoped, @NoneScoped e @CustomScoped
  - ⇒ tutorial JEE7: @ApplicationScoped, @SessionScoped, @FlowScoped, @RequestScoped e @Dependent
- ✓ @ManagedProperty
  - ⇒ permite injetar valores em uma property
- ✓ @Named (CDI): JEE 7
  - ⇒ JSF 2.2 recomenda seu uso

JEE 8: pacote javax.faces.bean será obsoleto

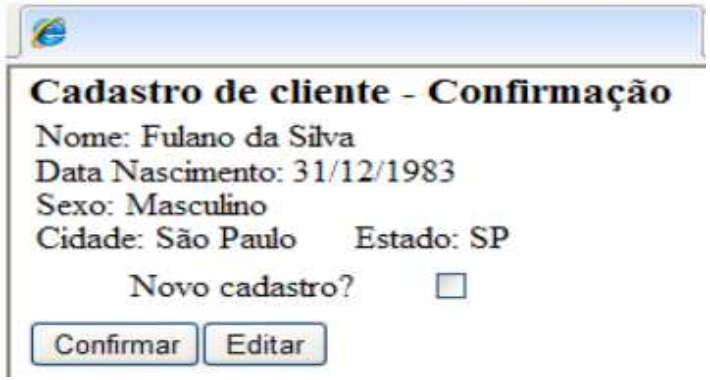
```
@ManagedBean(name="clienteBean", eager = true)
@ApplicationScoped
public class ClienteBean implements Serializable {
   @ManagedProperty(value="#{localidadeClienteBean}")
   private LocalidadeCliente localidade;
   public String criarCliente() {
      LocalidadeCliente localidadeCliente = new LocalidadeCliente();
      localidadeCliente.setCidade(localidade.getCidade());
      localidadeCliente.setEstado(localidade.getEstado());
      ...
      return "index";
   }
}
```

JavaServer Faces: Controle de navegação

- **Navegação explícita**
  - ✓ XML (faces-config)
  - ✓ elemento <navigation-rule>
    - ⇒ página origem
      - ▹ <from-view-id>
    - ⇒ regras
      - ▹ <navigation-case>
      - ▹ <from-outcome>
        - ▹ atributo **action/outcome**
          - ▹ estática: valor simples
          - ▹ dinâmica: método MBean
    - ⇒ página destino
      - ▹ <to-view-id>
    - ⇒ navegação condicional (JSF 2.0)
      - ▹ <if>

```
<navigation-rule>
  <from-view-id>/confirmacao.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>index</from-outcome>
    <if>#{clienteBean.novoCadastro}</if>
    <to-view-id>/cadastro.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>index</from-outcome>
    <to-view-id>/index.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>cadastro</from-outcome>
    <to-view-id>/cadastro.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

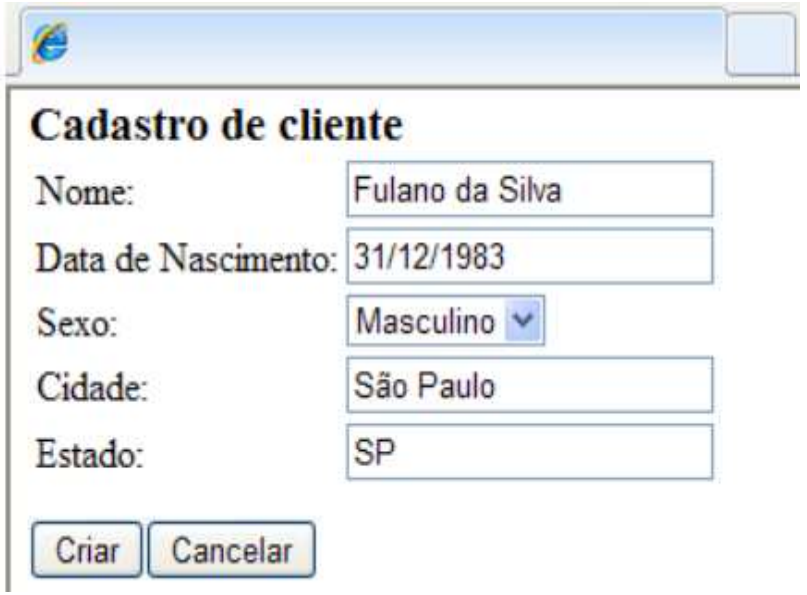
```
<h:selectBooleanCheckbox value="#{clienteBean.novoCadastro}" />
<h:commandButton value="Confirmar"
     action="#{clienteBean.criarCliente}" />
<h:commandButton value="Editar"
     action="cadastro" />
```



JavaServer Faces: Controle de navegação

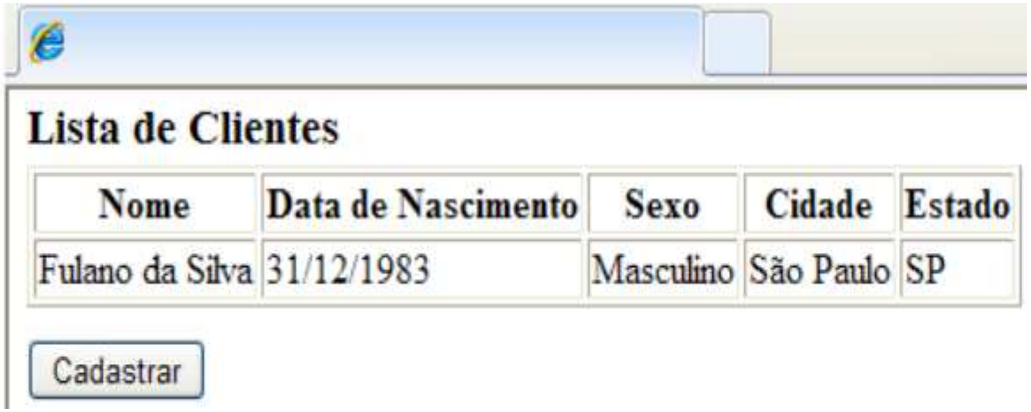
- **Navegação implícita (JSF 2.0)**
  - ✓ dispensa o uso de XML
    - ⇒ outcome é uma página web
      - atributo **outcome**
        - button, link e outputLink
        - não permite invocar métodos
      - atributo **action**
        - commandButton e commandLink

```
<h:commandButton value="Criar"
    action="#{clienteBean.criarCliente}" />
<h:commandButton value="Cancelar"
    action="index" />
```




```
public String criarCliente() {
   clienteDAO.salvar(cliente);
   listaCliente.add(cliente);

   return "index";
}
```



[33] FCC - 2010 - MPE-SE

No Java Server Pages (JSP), BackBean

a) não seguem a convenção JavaBean, portanto não possuem getters e setters.

b) são métodos que acionam os Action Event Handlers, sem procurar nenhuma regra de navegação para voltar a mesma página.

c) são classes simples, não herdam de ninguém nem são obrigados a implementar nenhuma interface.

d) são componentes associados à página, onde são criadas as views de primeiro acesso.

e) são processos que executam os validadores existentes na view, para aplicação das regras a todos os valores digitados.

[34] FCC - 2013 - TRT-SC

Considere as instruções abaixo encontradas em um arquivo de uma aplicação que utiliza JSF

```
<managed-bean>
<managed-bean-name>func</managed-bean-name>
<managed-bean-class>bean.Funcionario</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Essas instruções indicam a existência de um *bean* gerenciado (classe Funcionario.java) no pacote *bean* que poderá ser referenciado nas páginas JSP por meio da palavra func . O arquivo correto no qual essas instruções são colocadas é o

a) context.xml.
b) web-inf.xml.
c) web.xml.
d) faces-config.xml.
e) config - bean.xml.

[35] CESPE - 2009 - TCU

```
1<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
2<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
3<html>
4<body>
5     <div id="container">
6          <f:view>
7               <h:form>
8                    <table>
9                         <tr><td>Email</td>
10                             <td><h:inputText id="email" value="#{flight.email}"/></td></tr>
11                        <tr><td><h:outputText id="departure" value="#{flight.departure}"/></td> </tr>
12                        <tr><td><h:outputText value="#{booking_messages['city']}"/></td>
13                             <td><h:selectOneMenu value="#{flight.city}" required="true" /></td>
14                        </tr>
15                   </table>
16                   <h:commandButton type="submit" value="#{booking_messages['reserve']}"
17                        action="#{flight.reserve}"/>
18                   <h:messages/>
19              </h:form>
20         </f:view>
21    </div>
22</body>
23</html>
```

Na instrução ***value="#{flight.*}"***, o nome do *backing bean* é ***flight*** e o segundo valor representa o nome do campo definido no *bean.*

[36]

CESPE - 2013 - INPI

Quando registrado em JSF 2 (Java Server Faces), um *managed bean* permanece no escopo de *session.*

CESPE - 2012 - ANAC

A validação de dados de um componente pode ser uma das funções de um backing bean, em uma aplicação JSF.

CESPE - 2011 - BRB

No JavaServer Faces, para que as páginas de uma aplicação acessem as propriedades e operações de uma classe Bean, é necessário realizar um mapeamento da classe, que pode ser feito no arquivo faces-config.xml ou utilizando-se annotations

JavaServer Faces: Componentes UI

- ✓ geram a view
  - ⇒ arquivo composto por tags que geram uma árvore de componentes UI
  - ⇒ renderizada a visão da página web
- ✓ PDL ou VDL
  - ⇒ JEE 5 (JSF 1.2): **JSP**
    - ▹ JSF tag libraries html e core
    - ▹ JSTL (completo)
  - ⇒ JEE 6 (JSF 2.0): XHTML (**facelets**)
    - ▹ facelet e composite tag library
    - ▹ JSTL (function e core)
  - ⇒ JEE 7 (JSF 2.2)
    - ▹ tag libraries para HTML5

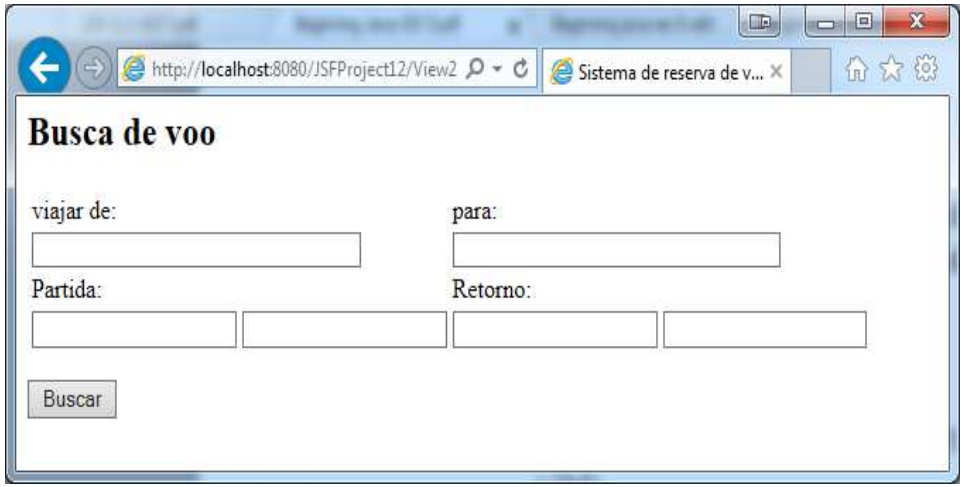
JSF HTML	http://java.sun.com/jsf/html	h:
JSF Core	http://java.sun.com/jsf/core	f:
JSF Facelets	http://java.sun.com/jsf/facelets	ui:
JSF Composite	http://java.sun.com/jsf/composite	composite:
HTML5	http://xmlns.jcp.org/jsf http://xmlns.jcp.org/jsf/passthrough	jsf: p:

▷ Kits de componentes UI

- ⇒ Mojarra UI (implementação de referência)
- ⇒ MyFaces (Apache)
- ⇒ RichFaces (JBoss)
- ⇒ Ajax4JSF (JBoss)
- ⇒ PrimeFaces

JavaServer Faces: Componentes UI

- **JSP view**
  - ✓ JEE 5 utiliza páginas JSP 2.1
    - ⇒ JSF tag libraries html e core
    - ⇒ JSTL (suporte completo)
    - ⇒ Unified EL
  - ✓ considerado obsoleto para JEE 6
    - ⇒ problema de performance
  - ✓ árvore de componentes UI
    - ⇒ tag <f:view>
      - ▹ tag raiz para elementos JSF




```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>

<html>
<f:view>
  <head> <title>Sistema de reserva de voos</title>
  </head>
  <body>
    <h:form>
      <h2>Busca de voo</h2>
      <table>
        <tr>
          <td>viajar de:</td>     <td>para:</td>
        </tr>
        <tr>
          <td><h:inputText value="#{flight.origination}"/></td>
          <td><h:inputText value="#{flight.destination}"/></td>
        </tr>
        <tr><td>Partida:</td>     <td>Retorno:</td></tr>
        <tr>
          <td><h:inputText value="#{flight.departDate}"/></td>
          <td><h:inputText value="#{flight.departTime}"/></td>
          <td><h:inputText value="#{flight.returnDate}"/></td>
          <td><h:inputText value="#{flight.returnTime}"/></td>
        </tr>
      </table>
      <p>  <h:commandButton value="Buscar" action="submit" /></p>
    </h:form>
  </body>
</f:view>
</html>
```

JavaServer Faces: Componentes UI

▷ HTML Tag Library

✓ fornece componentes padrões para páginas HTML

⇒ tags são renderizadas em instâncias de componentes UI

⇒ diretiva: `<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>`

⇒ namespaces:
```
xmlns:h="http://java.sun.com/jsf/html"
xmlns:h="http://xmlns.jcp.org/jsf/html"
```

h:body	<body>	h:inputFile	<input type=file>	h:panelGrid	<table> com <tr> e <td>
h:button h:commandButton	<input type=submit>	h:inputHidden	<input type=hidden>	h:panelGroup	ou
		h:inputSecret	<input type=password>	h:panelpassthrough.Element	
h:commandLink h:link h:outputLink	<a href>	h:inputText	<input type=text>	h:selectBooleanCheckbox h:selectManyCheckbox	<input type=checkbox>
		h:inputTextarea	<textarea>		
h:dataTable	<table>	h:message h:messages		h:selectManyListbox h:selectManyMenu h:selectOneListbox h:selectOneMenu	<select>
h:column		h:outputFormat	ou texto plano		
h:doctype	<!DOCTYPE>	h:outputLabel	<label>		
h:form	<form>	h:outputScript	<script>	h:selectOneRadio	<input type=radio>
h:graphicImage	<img>	h:outputStylesheet	<link>		
h:head	<head>	h:outputText	ou texto plano		

[37] FUMARC - 2011 - BDMG

Em relação aos conceitos da tecnologia *JavaServer Faces* (JSF), analise as seguintes afirmativas:

I. JSF fornece um conjunto de componentes de interface de usuário – componentes JSF – que ajudam na construção de páginas Web.

II. Os componentes JSF podem ser adicionados a páginas JSP por meio das bibliotecas de *tags* personalizadas *(tag libraries*).

III. Além dos componentes básicos, existem bibliotecas nativas de componentes JSF adaptados para interfaces Swing e AWT, por exemplo.

Marque a alternativa CORRETA:

a) apenas I e II são verdadeiras.
b) apenas I e III são verdadeiras.
c) apenas II e III são verdadeiras.
d) todas são verdadeiras.

JavaServer Faces: Componentes UI

▷ Core Tag Library

✓ fornece ações predefinidas independentes de RenderKit específico

⇒ tratamento de eventos, validação, conversão, etc.

⇒ diretiva: `<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>`

⇒ namespaces:
```
xmlns:f="http://java.sun.com/jsf/core"
xmlns:f="http:// xmlns.jcp.org /jsf/core"
```

eventos	f:actionListener f:phaseListener f:setPropertyActionListener f:valueChangeListener f:event
validação	f:validateBean f:validateDoubleRange f:validateLength f:validateLongRange f:validateRegex f:validateRequired f:validator
conversão	f:convertDateTime f:converter f:convertNumber
outras ações	f:ajax f:selectItem f:selectItems f:view f:subview f:viewAction f:viewParam f:attribute f:attributes f:passThroughAttribute f:passThroughAttributes f:param f:resetValues f:verbatim f:metadata f:facet f:loadBundle

JavaServer Faces: Componentes UI

▷ Core Tag Library: Validadores

✓ executados em fase específica no ciclo de vida (process validations)

⇒ implementa a interface javax.faces.validator.Validator

⇒ possuem padrões de mensagens de erro associados

✓ formas de validar valores de componentes

⇒ tag do validador dentro do componente

```
<h:inputText value="#{bookController.book.price}">
   <f:validateLongRange minimum ="1"  maximum="500" />
</h:inputText>
```

⇒ validador no atributo validator

```
<h:inputText value="#{book.isbn}" validator="isbnValidator" />
```

⇒ tag f:validator com o validador no atributo validatorId

```
<h:inputText value="#{book.isbn}">
  <f:validator validatorId="isbnValidator" />
</h:inputText>
```

JavaServer Faces: Componentes UI

▷ Core Tag Library: Conversores

- ✓ executados em fase específica no ciclo de vida (apply request values)
  - ⇒ implementa a interface javax.faces.convert.Converter
  - ⇒ possuem padrões de mensagens de erro associados
- ✓ automático para properties com tipos primitivos
- ✓ Conversores padrões de tipos comuns
  - BigDecimalConverter
  - BigIntegerConverter
  - BooleanConverter
  - ByteConverter
  - CharacterConverter
  - DateTimeConverter
  - DoubleConverter
  - EnumConverter
  - FloatConverter
  - IntegerConverter
  - LongConverter
  - NumberConverter
  - ShortConverter
- ✓ tags predefinidas `<f:convertDateTime>` e `<f:convertNumber>`
- ✓ formas de converter valores de componentes

```
<h:inputText converter="javax.faces.convert.IntegerConverter" />
```

```
<h:inputText value="#{loginBean.age}" />
  <f:converter converterId="Integer" />
</h:inputText>
```

```
<h:outputText value="#{cart.total}">
  <f:convertNumber currencySymbol="$" type="currency" />
</h:outputText>
```

[38] CESPE - 2012 - TJ-AC

A inclusão de ***inputText*** em uma página JSF permite validar o tamanho mínimo dos valores digitados por meio da utilização do seguinte código:

```
<h:inputText id="cpf" label="CPF" value="#{UsuarioBean.cpf}">
      <f:validateLongRange
            minimum="#{UsuarioBean.minimum}"
            maximum="#{UsuarioBean.maximum}" />
</h:inputText>
```

JavaServer Faces: Eventos

- **Eventos JSF**
  - PhaseEvent
    - ⇒ ocorrem no início e fim de cada fase do ciclo de vida JSF
    - ⇒ interface PhaseListener

```
public void afterPhase(PhaseEvent event);
public void beforePhase(PhaseEvent event);
public PhaseId getPhaseId();
```

    - ⇒ associa-se a
      - ⇒ instância de Lifecycle (todas as páginas)
        - ⇒ registrado no faces-config.xml

```
<lifecycle>
    <phase-listener>listeners.AuthorizationListener</phase-listener>
</lifecycle>
```

      - ⇒ UIViewRoot (página específica)
        - ⇒ registrado na página web

```
<h:commandButton value="Confirmar"
    action="#{clienteBean.criarCliente}">
    <f:phaseListener type="listeners.AuthorizationListener" />
</h:commandButton>
```

JavaServer Faces: Eventos

- **Eventos JSF**
  - FacesEvent
    - ⇒ são disparados pelos componentes UI
    - Action Events
      - ⇒ componentes de ação (clique do mouse)
        - ⇒ CommandButton ou CommandLink
      - ⇒ Apply Request Values (true) ou Invoke Application (false)
    - Value Change Events
      - ⇒ mudança em componentes que armazenam valores
        - ⇒ ValueHolder: outputText
        - ⇒ EditableValueHolder: inputText ou selectOneMenu
      - ⇒ Apply Request Values (true) ou Process Validations (false)

```
<h:inputText id="nome" value="#{clienteBean.nome}" onchange="submit()"
        valueChangeListener="#{clienteBean.nomeChagend}" />

<h:commandButton value="Confirmar" action="#{clienteBean.criarCliente}" />
<h:commandButton value="Fatorial" actionListener="#{calc.findFatorial}" />

<h:commandButton value="Fatorial">
        <f:actionListener type="listeners.CalcActionListener" />
</h:commandButton>
```

JavaServer Faces: Eventos

- **Eventos do sistema (JSF 2.0)**
  - Application-level (SystemEvent)

PostConstructApplicationEvent (startup)
PreDestroyApplicationEvent (shutdown)
ExceptionQueuedEvent

⇒ registrado no faces-config.xml

```
<application>
   <system-event-listener>
      <system-event-class>javax.faces.event.PostConstructApplicationEvent</system-event-class>
      <system-event-listener-class>listeners.ListingLoader</system-event-listener-class>
   </system-event-listener>
</application>
```

  - Component-level (ComponentSystemEvent)

PreRenderComponentEvent	PreValidateEvent	PostRestoreStateEvent
PreRenderViewEvent	PostValidateEvent	PostConstructViewMapEvent
PostAddToViewEvent		PreDestroyViewMapEvent

⇒ registrado no próprio componente (f:event)

```
<h:inputText id="userName" value="#{person.name}">
        <f:event type="postValidate" listener="#{person.checkName}" />
</h:inputText>
```

```
public void checkName(ComponentSystemEvent csEvent) {
        UIComponent component = csEvent.getComponent();
... }
```

[39] CESPE - 2012 - TJ-AC

Acerca de sistemas transacionais, julgue os itens seguintes.

A tecnologia JSF suporta eventos de mudança de valores por meio da seleção de um link.

JavaServer Faces: Componentes UI

▷ Ajax

- ✓ JSF 2.0 (JEE 6) provê suporte nativo a Ajax
  - ⇒ atualizar partes da aplicação, sem recarregar a página inteira
  - ⇒ validar componentes individuais
- ✓ formas de adicionar Ajax em aplicações JSF
  - ⇒ **tag <f:ajax>**

```
<h:inputText id="inputname"
      value="#{userBean.name}" />
<h:outputText id="outputname"
      value="#{userBean.name}" />
<h:commandButton id="submit"
      value="Submit">
   <f:ajax event="click"
      execute="inputname"
      render="outputname"/>
</h:commandButton>
```

```
<f:ajax>
   <h:inputText id="user"
       value="#{user.name}" />
   <h:inputSecret id="password"
       value="#{user.password}"/>
   <h:outputText id="result"
       value="#{user.result}" />
   <h:commandButton id="submit"
       value="Logar" />
</f:ajax>
```

  - ⇒ **JavaScript Library (jsf.js)**

```
<h:outputScript name="jsf.js" library="javax.faces" target="head">
<h:inputText id="inputname" value="#{userBean.name}" />
<h:outputText id="outputname" value="#{userBean.name}" />
<h:commandButton id="submit" value="Submit"
    onclick="jsf.ajax.request(this, event,
            {execute:'inputname',render:'outputname'});
            return false;" />
```

[40]

CESPE - 2014 - TJ-SE

Em aplicações *web* nos padrões da JSF, é possível utilizar recursos Ajax para criar páginas dinâmicas, como, por exemplo, por meio da *tag* f:ajax, conforme apresentado na sintaxe abaixo.

```
<h:inputText value="#{bean.message}">
        <f:ajax />
</h:inputText>
```

CESPE - 2010 - TCU

Para suportar a construção de aplicações com Ajax e JSF, recomenda-se aos desenvolvedores de páginas que usem a *tag* <f:ajax>, relacionada ao processamento de pedidos http assíncronos.

JavaServer Faces: Componentes UI

- **Facelets (XHTML)**
  - ✓ VDL padrão JEE 6 (JSF 2.0)
    - ⇒ JSF tag libraries html e core
    - ⇒ facelet tag library
    - ⇒ JSTL (function e core)
    - ⇒ JEE 7 (JSF 2.2)
      - tag libraries para HTML5
  - ✓ suporte a templates HTML
    - ⇒ reuso de estrutura visual
    - ⇒ facelet é baseado em composições
  - ✓ é um documento XHTML
    - ⇒ conformidade com DTD transacional

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

  - ✓ Diferenças com JSP
    - ⇒ melhor performance
    - ⇒ menor tempo de compilação
      - componentes UI gerados em estrutura de árvore

JavaServer Faces: Componentes UI

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>

<html>
<f:view>
  <head> <title>Sistema de reserva de voos</title></head>
  <body>
    <h:form>
      <h2>Busca de voo</h2>
      <table>
        <tr><td>viajar de:</td>   <td>para:</td></tr>
        <tr>
          <td><h:inputText value="#{flight.origination}"/></td>
          <td><h:inputText value="#{flight.destination}"/></td>
        </tr>
        <tr><td>Partida:</td>     <td>Retorno:</td></tr>
        <tr>
          <td><h:inputText value="#{flight.departDate}"/></td>
          <td><h:inputText value="#{flight.departTime}"/></td>
          <td><h:inputText value="#{flight.returnDate}"/></td>
          <td><h:inputText value="#{flight.returnTime}"/></td>
        </tr>
      </table>
      <p><h:commandButton value="Buscar" action="submit" /></p>
    </h:form>
  </body>
</f:view>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html  xmlns="http://www.w3.org/1999/xhtml"
       xmlns:h="http://xmlns.jcp.org/jsf/html"
       xmlns:f="http://java.sun.com/jsf/core">

<h:head><title>Sistema de reserva de voos</title></h:head>
<h:body>
  <h:form>
    <h2>Busca de voo</h2>
    <table>
      <tr><td>viajar de:</td>     <td>para:</td></tr>
      <tr>
        <td><h:inputText value="#{flight.origination}"/></td>
        <td><h:inputText value="#{flight.destination}"/></td>
      </tr>
      <tr><td>Partida:</td>  <td>Retorno:</td></tr>
      <tr>
        <td><h:inputText value="#{flight.departDate}"/></td>
        <td><h:inputText value="#{flight.departTime}"/></td>
        <td><h:inputText value="#{flight.returnDate}"/></td>
        <td><h:inputText value="#{flight.returnTime}"/></td>
      </tr>
    </table>
    <p><h:commandButton value="Buscar" action="submit" /></p>
  </h:form>
</h:body>
</html>
```

JavaServer Faces: Componentes UI

▷ Facelets Tag Library

- ✓ Facelet baseado em composições
- ✓ fornece componentes para criação de templates
  - ⇒ definição de menus, estrutura da página, etc
  - ⇒ namespaces: `xmlns:ui="http://java.sun.com/jsf/facelets"`
    `xmlns:ui="http://xmlns.jcp.org/jsf/facelets"`
  - ⇒ tags

ui:component	ui:define	ui:include	ui:repeat
ui:composition	ui:decorate	ui:insert	ui:remove
ui:debug	ui:fragment	ui:param	

- ✓ Aplicação Facelet
  - ⇒ páginas XHTML
    - ▹ Template page
      - ▹ **<ui:insert>** para definir pontos de reuso
      - ▹ **<ui:include>** para inserir componentes padrões
    - ▹ Compositions pages
      - ▹ **<ui:composition>** informa o template
      - ▹ **<ui:define>** insere conteúdo nos pontos de reuso

JavaServer Faces: Componentes UI

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
<h:head>
  <title><ui:insert name="title" /></title>
</h:head>
<h:body>
  <table>
    <tr><td colspan="2">
        <ui:insert name="header">
          <ui:include src="Header.xhtml" />
        </ui:insert>
    </td></tr>
    <tr>
      <td>
        <ui:insert name="menu">
          <ui:include src="Menu.xhtml" />
        </ui:insert>
      </td>
      <td>
        <ui:insert name="principal" />
      </td>
    </tr>
    <tr><td colspan="2">
      <ui:insert name="footer">
        <ui:include src="Footer.xhtml" />
      </ui:insert>
    </td></tr>
  </table>
</h:body>
</html>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:h="http://xmlns.jcp.org/jsf/html"
                xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
       <h:outputLabel value="Cabeçalho" />
</ui:composition>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:h="http://xmlns.jcp.org/jsf/html"
                xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
  <h:form>
    <h2>Cadastro</h2>
    <ul>
      <li><h:commandLink value="Cliente" /></li>
      <li><h:commandLink value="Fornecedor" /></li>
      <li><h:commandLink value="Produtor" /></li>
    </ul>
  </h:form>
</ui:composition>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:h="http://xmlns.jcp.org/jsf/html"
                xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
       <h:outputLabel value="Rodapé" />
</ui:composition>
```

JavaServer Faces: Componentes UI

```
<ui:composition template="Template.xhtml"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  xmlns:h="http://xmlns.jcp.org/jsf/html">

  <ui:define name="title">#{"Cadastro Fornecedores"}</ui:define>

  <ui:define name="principal">
    <h:outputLabel value="Cadastro Fornecedores" />
    <h:form>
      <h:inputHidden value="#{fornecedorMB.id}" id="idFornecedor" />
      <h:panelGrid columns="2">
        <h:outputLabel value="Nome" for="iNome" />
        <h:inputText value="#{fornecedorMB.nome}" id="iNome" />
        <h:outputLabel value="Cnpj" for="iCnpj" />
        <h:inputText value="#{fornecedorMB.cnpj}" id="iCnpj" />
      </h:panelGrid>
      <h:messages showDetail="true" showSummary="true" />
      <h:commandButton action="#{fornecedorMB.doSalvar}" value="Salvar" />
      <h:commandButton action="#{fornecedorMB.doExcluir}" value="Excluir" />
    </h:form>
  </ui:define>
</ui:composition>
```

[41]

CESPE - 2010 - TCU

No desenvolvimento de conteúdos para apresentação, o uso de *facelets* traz vantagens em relação ao uso de JSP. Uma delas é a maior modularidade, com o uso de *templates* e componentes compostos *(composite)*.

CESPE - 2013 - CPRM

Facelets são utilizadas para desenvolver visões (views) JavaServer Faces (JSF) com linguagem HTML e XHTML, em conformidade com a *transitional document type definition*, sendo, ainda, compatível com a biblioteca de *tag* JSF.

CESPE - 2014 - TJ-SE

É possível utilizar XHTML no desenvolvimento de *facelets* para criar páginas *web* compatíveis com a JSF (*JavaServer Faces*) para apresentação dos dados. Na versão Java EE 7, essa forma de apresentação é mais indicada que a JSP (*JavaServer Pages*), uma vez que esta não suporta todos os novos recursos da versão Java EE 7.

[42] FCC - 2012 - TST

Para criar as páginas XHTML de uma aplicação JSF é possível utilizar um conjunto de bibliotecas de tags JSF. Algumas dessas bibliotecas são HTML, *Core e Facelets.* Considere os fragmentos de códigos abaixo, que utilizam *tags* dessas bibliotecas:

Fragmento de código **I**:

```
<h1>
   <ui:insert name="titulo">
      Título
   </ui:insert>
</h1>
```

Fragmento de código **II**:

```
<h:inputText id="usuário"
   value="#{usuarioBean.usuario.nome}"/>
```

Fragmento de código **III**:

```
<h:selectOneMenu id="lista">
   <f:selectItems
      value="#{optionBean.optionList}">
   </f:selectItem>
</h:selectOneMenu>
```

A correlação correta entre o fragmento de código e a biblioteca de *tags* utilizada é

a) I-*Facelets*, II-HTML e III-Core.

b) I-Core, II-*Facelets*, e III-HTML.

c) I-HTML, II-*Core*, e III-Facelets.

d) I-HTML, II-*Facelets*, e III-HTML.

e) I-HTML, II-HTML, e III-Core.

Java API for JSON Processing (JSON-P)

▷ JavaScript Object Notation (JSON)

- ✓ formato para troca de dados
- ✓ utilizado em web services Restful, Ajax e BD NoSQL
- ✓ estrutura de dados
  - ⇒ Objetos, Arrays e valores

```
{
   "firstName": "Duke",
   "lastName": "Java",
   "age": 18,
   "streetAddress": "100 Internet Dr",
   "city": "JavaTown",
   "state": "JA",
   "postalCode": 12345,
   "phoneNumbers": [
      {"Mobile": "111-111-1111"},
      {"Home": "222-222-2222"}
   ]
}
```

▷ JSON-P (JSR 353) : JEE 7

- ✓ gera e faz parsing de dados JSON
- ✓ não é JSONP (JSON with padding)
- ✓ estrutura da API
  - ⇒ **Object model (alto nível)**
    - ▹ semelhante ao DOM
    - ▹ cria a árvore de dados JSON
    - ▹ pacote javax.json
    - ▹ principal componente: Json
    - ▹ JsonObject, JsonArray e JsonValue
    - ▹ criação de objetos: padrão builder ou JsonReader
    - ▹ escrita: JsonWriter

Java API for JSON Processing (JSON-P)

- **JSON-P (JSR 353) : JEE 7**
  - ✓ estrutura da API
    - ⇒ **Streaming model (baixo nível)**
      - parser de leitura baseado em evento
      - semelhante ao StAX
      - pacote javax.json.stream
        - JsonParser: acesso de leitura (pull parser)
        - JsonGenerator: escreve no fluxo
  - Object Model X Streaming Model

Objeto	Streaming
DOM	StAX
alto-nível	baixo-nível
push parser	pull parser
object-tree (memory)	event-based (streaming)
acesso aleatório	acesso sequencial

  - JEE 6: JAX-RS + JAXB
    - ⇒ terceiros : parser JSON e mapeamento JSON/XML
  - JEE 8: JSON-B [+ JAX-RS]

Java API for JSON Processing (JSON-P)

```
URL url = new  URL("https://graph.facebook.com/search?q=java&type=post");
try (   InputStream input = url.openStream();
        JsonReader reader = Json.createReader(input)) {
            JsonObject object = reader.readObject();
            JsonArray results = object.getJsonArray("data");
            for (JsonObject result : results.getValuesAs(JsonObject.class)) {
                System.out.print(result.getJsonObject("from").getString("name"));
                System.out.print(": ");
                System.out.println(result.getString("message", ""));
                System.out.println("----------");
            }
        }
```

```
URL url = new URL("https://graph.facebook.com/search?q=java&type=post");
try (   InputStream input = url.openStream();
        JsonParser parser = Json.createParser(input)) {
            while (parser.hasNext()) {
                Event e = parser.next();
                if (e == Event.KEY_NAME) {
                    switch (parser.getString()) {
                    case "name":
                        parser.next();
                        System.out.print(parser.getString());
                        System.out.print(":  ");
                        break;
                    case "message":
                        parser.next();
                        System.out.println(parser.getString());
                        System.out.println("---------");
                        break;
                    }
                }
            }
        }
```

[43] CESPE - 2014 - ANATEL

Na plataforma JEE (Java Enterprise Edition) versão 6, não é possível encontrar bibliotecas da própria plataforma para o consumo dos serviços REST no formato JSON.

Java API for WebSocket

- **protocolo websocket**
  - ✓ fornece canal de comunicação full-duplex sobre TCP
  - ✓ parte do HTML5 initiative
    - ⇒ W3C: especificação da API (draft)
    - ⇒ IETF: especificação do protocolo (RFC 6455)
  - ✓ estrutura
    - ⇒ handshake
      - ▹ inicia com uma conexão HTTP
    - ⇒ transferência de dados
      - ▹ texto ou binário

```
GET ws://echo.websocket.org/endpoint HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: echo.websocket.org
Origin: http://websocket.org
Sec-WebSocket-Key: uRovscZjNol/umbTt5uKmw==
Sec-WebSocket-Version: 13
```

```
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Accept: rLHCkwO9GAH/ZSFhBATDKrU=
```

- **WebSocket (JSR 356) : JEE 7**
  - ✓ permite manipular endpoints
  - ✓ pacotes: `javax.websocket.server` e `javax.websocket`
  - ✓ WebSocket Endpoint
    - ⇒ instância de javax.websocket.Endpoint
    - ⇒ pode ser programático ou por anotations
    - ⇒ uma instância por conexão (!= servlet)

Gabarito

[01]	C-C
[02]	E
[03]	C
[04]	E
[05]	D
[06]	C-E
[07]	E-C
[08]	D
[09]	A
[10]	C
[11]	B

[12]	D
[13]	E
[14]	E-C
[15]	C
[16]	C-E
[17]	C-E
[18]	B
[19]	A
[20]	E
[21]	C
[22]	C-C-E

[23]	B
[24]	C-E
[25]	C-C
[26]	E
[27]	C
[28]	C-C
[29]	C
[30]	B
[31]	A
[32]	E
[33]	C

[34]	D
[35]	C
[36]	E-C-C
[37]	A
[38]	C
[39]	E
[40]	C-C
[41]	C-C-C
[42]	A
[43]	C

ITnerante

Java Enterprise Edition módulo III

PROVAS DE TI
TUDO PARA VOCÊ PASSAR

Leonardo Marcelino
http://www.itnerante.com.br/profile/LeonardoMarcelino