

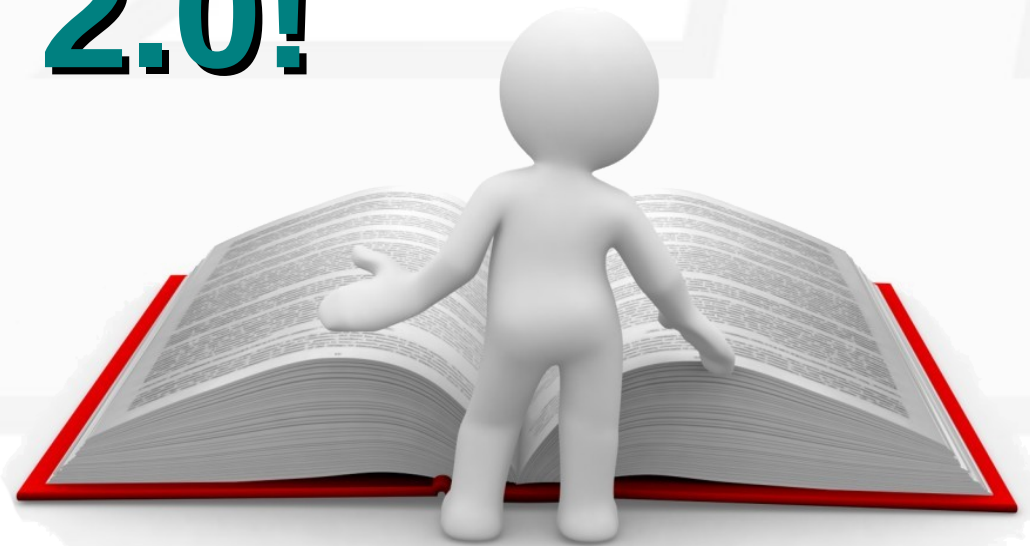


**Cursos, soluções e serviços baseados
em software livres e padrões abertos
para ambientes de missão crítica**

- *Experiência em missão crítica de missão crítica*
- *Pioneira no ensino de Linux à distância*
- *Parceira de treinamento IBM*
- *Primeira com LPI no Brasil*
- *+ de 30.000 alunos satisfeitos*
- *Reconhecimento internacional*
- *Inovação com Hackerteen e Boteconet*



As grandes novidades do JSF 2.0!



- JSF 1.0 – Lançado em março de 2004, sua especificação é a JSR 127, não faz parte do conjunto de tecnologias padrões do Java EE. Compatível com servlet 2.3 e JSP 1.2
- JSF 1.1 – Lançado em maio de 2004, bug fixes corrigidos e ganhos de performance, a especificação continua a mesma, continua não fazendo parte do padrão Java EE, possui suporte para a JDK 1.3 ou superior. Compatível com servlet 2.3 e JSP 1.2.

- JSF 1.2 – Lançado em maio de 2006, compatível com servlet 2.5 e JSP 2.1. Segue a JSR 252. Faz parte do Java EE 5. Possui várias vantagens por utilizar JSP 2.1, por exemplo, integração com JSTL e Unified Expression Language, entre outras.

Principais mudanças e melhorias do JSF 2.0



- Lançado em Julho de 2009. Desenvolvido seguindo a JSR 314. Faz parte do Java EE 6. Compatível com CDI, Servlet 3, JSP 2.2.
- Inúmeras mudanças foram adotadas na versão 2.0. Para acompanhar todas as mudanças consulte o preface 1 da JSR 314 liberado em julho de 2009
- As principais mudanças e melhorias adotadas foram:
 - ✓ AJAX nativo
 - ✓ Anotações
 - ✓ Navegação implícita e condicional

Principais mudanças e melhorias do JSF 2.0



- Integração nativa com a JSR 303 Bean Validation
- ✓ View Parameters
- ✓ Passagem de parâmetros via GET
- View Scope
- Custom Scope
- faces-config opcional
- Facelets como view padrão
- Project Stage
- Resources
- SelectItems mais “usáveis”
- ✓ Inúmeras outras melhorias

- Como era até o JSF 1.2

```
<link href=
"#{facesContext.externalContext.requestContextPath}/css/
estilo.css" rel="stylesheet">
```

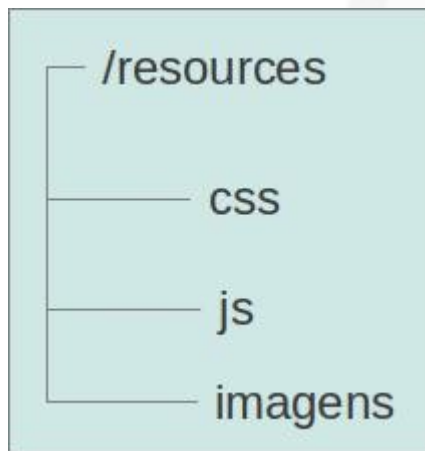
- No JSF 2 novas tags foram adicionadas para o carregamento de CSS, Javascript, imagens
- Pode ser utilizado um mecanismo padrão para carregar estes recursos na página, que consiste em criar um diretório chamado resources na raiz da aplicação
- É possível definir se o javascript ou css será carregado no head ou body da página através do atributo target


```
<h:outputStylesheet library="css" name="meucss.css"/>
```

```
<h:outputScript library="js" name="meujs.js"  
target="head"/>
```

```
<h:graphicImage library="imagens" name="logo.jpg"/>
```

- #{resources['images:logo.jpg']}
- Exemplo de estrutura de diretórios:



- Com o JSF 2 é possível criar managed beans e atribuir um escopo, converters e validators com anotações sem utilizar nenhuma linha de XML no faces-config.
- Abaixo algumas anotações incluídas no JSF 2
`@ManagedBean`, `@ManagedProperty`
`@FacesValidator`, `@FacesConverter`
`@ApplicationScoped`, `@SessionScoped`, `@RequestScoped`,
`@ViewScoped`, `@NoneScoped`, `@CustomScoped`
- O faces-config.xml pode ser utilizado apenas para regras de navegação mais complexas.

- Nas versões 1.x do JSF é necessário recorrer a frameworks JSF de terceiros, por exemplo, RichFaces, IceFaces, PrimeFaces.
- Abaixo um trecho de código utilizando JSF 1.2

```
<h:form id="formulario">
    <h:panelGrid columns="2">
        <h:inputText id="inputNome" value="#{usuarioMB.nome}">
            <a4j:support event="onkeyup" reRender="outNome" />
        </h:inputText>
        <h:outputText id="outNome" value="#{usuarioMB.name}" />
    </h:panelGrid>
</h:form>
```

- Uma das principais melhorias no JSF 2 é a possibilidade de fazer requisições de forma nativa sem a necessidade de um utilizar um framework visual de terceiros para isso.
- Abaixo o mesmo trecho de código utilizando JSF 2.0

```
<h:form id="formulario">
    <h:panelGrid columns="2">
        <h:inputText id="inputNome" value="#{usuarioMB.nome}">
            <f:ajax event="keyup" render="outNome"/>
        </h:inputText>
        <h:outputText id="outNome" value="#{usuarioMB.nome}" />
    </h:panelGrid>
</h:form>
```

- Nova API Javascript para requisições Ajax e acesso aos componentes:

```
<h:outputScript name="jsf.js" library="javax.faces"
target="head"></h:outputScript>
```

- jsf.specversion
- jsf.getProjectStage()
- jsf.ajax.addOnEvent(metodoDeCallback) / addOnError

-

```
<h:commandButton id="button1" action="addComment"
value="Comentar" onclick="jsf.ajax.request(this, event,
{execute:'button1', render: 'status', onevent:
handleEvent, onError: handleError}); return false;"/>
```

- @all - Todos os components da página.
- @none – Nenhum componente da página.
- @this – O selemento que disparou o pedido.
- @form – O Formulário e os components desse formulário.

- Com o JSF 2 novos escopos foram adicionados: `@ViewScoped` e `@CustomScoped`
- Utilizando CDI em substituição aos Managed Beans podemos utilizar o escopo de conversação: `@ConversationScoped`
- Para quem já estava acostumado a utilizar o JBoss Seam, o `@ViewScoped` é similar ao escopo de página e o `@ConversationScoped` similar ao escopo de conversação, já existentes desde a primeira versão do Seam.

View Scope

```
@ManagedBean(name="clienteMB")  
@ViewScoped  
public class ClienteMB {  
    ...  
}
```

```
@Named(value="clienteMB")  
@ConversationScoped  
public class ClienteMB {  
    ...  
}
```


- No JSF 1.2 as regras de navegação obrigatoriamente precisavam ser configuradas no faces-config.xml

```
<navigation-rule>
  <from-view-id>/usuario/boasvindas.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>new</from-outcome>
    <to-view-id>/usuario/cadastrousuario.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

- Com o JSF 2 basta apenas informar o caminho da view nas actions, eliminando a necessidade de criar todas as vezes regras de navegações no faces-config.xml

```
public String novoUsuario() {  
    usuario = new Usuario();  
    return "/usuario/cadastrousuario";  
}
```

- Também podemos fazer da seguinte forma:

```
<h:commandButton action="/usuario/cadastrousuario"  
value="Cadastrar usuário"
```

- faces-redirect=true indica um redirecionamento do navegador.
 - /user/list.xhtml?faces-redirect=true
- Inclui a extensão .xhtml caso não tenha.
- Inclui / caso não tenha.
- Procura por uma view com esse viewId, por exemplo:
 - user/list => /user/list.xhtml
- Caso não encontre a view, a navegação implícita é encerrada, e torna a string um outcome.
-

- Diferentemente da navegação implícita, a navegação condicional é definida no faces-config.xml
- Oferece maior poder de controle para a tag <navigation-case>
- Feita através da tag <if> dentro de um <navigation-case>
- Podem ser adicionadas quantas tags <if> forem necessárias
- A tag <if> deve apontar para um método ou expressão que retorne um valor booleano. A regra de navegação somente será executada caso todas as tags <if> retornem true.

Navegação condicional

```
<navigation-rule>
  <from-view-id> boasvindas.xhtml </from-view-id>
  <navigation-case>
    <from-outcome> edit </from-outcome>
    <if> #{usuarioBean.isAdmin} </if>
    <if> #{usuarioBean.idade > 18} </if>
    <to-view-id> cadastrousuario.xhtml </to-view-id>
  </navigation-case>
</navigation-rule>
```

- JSF 2 é compatível com a JSR 303 – Bean Validation
- É necessário que o ambiente ofereça suporte para o Bean Validation
- Possui mecanismos avançados e customizáveis de validação
- Proporciona maior reaproveitamento de código para o JSF pois as regras de validação não ficam vinculadas diretamente nas páginas

Bean Validation

```
public class Cliente implements Serializable {  
    @NotNull  
    private Long id;  
    @NotNull  
    private String nome;  
    @Size(max=30)  
    private String sobrenome;  
    @Past  
    private Date dataNascimento;  
    @Pattern(regex = ".*@.*\\.([a-z])+" message="O e-mail informado não  
    é válido")  
    private String email;  
    ...  
}
```

Bean Validation

```
<h:messages />

<h:form>

    <h:panelGrid columns="2">
        Nome <h:inputText value="#{clienteMB.cliente.nome}" />
        Sobrenome <h:inputText
            value="#{clienteMB.cliente.sobrenome}" >
            <f:validateBean disabled="true"/>
        - </h:inputText>
        -
        Email <h:inputText value="#{clienteMB.cliente.email}" />
        Id <h:inputText value="#{clienteMB.cliente.id}" />
        <h:commandButton action="page2" value="Enviar" />
    </h:panelGrid>
</h:form>
```


- View Parameters servem para atribuir os valores da query string passados por uma requisição GET de forma apropriada para o bean (managed bean ou CDI)
- Utilizamos as tags `<f:metadata>` e `<f:viewParam>` para atribuir os valores da query string no bean

```
<f:metadata>
```

```
    <f:viewParam name="id"
    value="#{clienteMB.cliente.id}"/>
```

```
</f:metadata>
```

-
- `Http://localhost:8080/client/show.xhtml?id=1`

Utilizamos as tags `<h:link>` ou `<h:button>` para requisições GET

```
<h:link outcome="editar" value="Editar">  
    <f:param name="id" value="#{cliente.id}"></f:param>  
</h:link>
```

- View Parameters podem ser utilizados para criar bookmarked links.

- No JSF 1.2 para utilizarmos componentes do tipo select one e select many de forma dinâmica era necessário ou criar uma lista de SelectItems ou um Map

```
public List<SelectItem> getComboStatus(){  
    List<SelectItem> lista = new ArrayList<SelectItem>();  
    List<Status> resultado = getBuscaStatus();  
  
    for(Status s: resultado)  
        lista.add(new SelectItem(s.id, s.nome));  
  
    return lista;  
}
```

```
<h:selectOneRadio value="#{produto.status}">
    <f:selectItems value="#{produto.comboStatus}" />
</h:selectOneRadio>
```

- Com o JSF 2 não precisamos mais criar uma lista de SelectItems
- É possível passar um List para os componentes do tipo select one e select many
- Novos atributos foram adicionados na tag <f:selectItems>

```
<h:selectOneRadio value="#{produto.status}">
    <f:selectItems value="#{produto.comboStatus}" var="s"
    itemLabel="#{s.nome}" itemValue="#{s.id}" />
</h:selectOneRadio>
```

- Com o JSF 2 é possível especificar no web.xml em quase fase do ciclo de desenvolvimento a aplicação se encontra

```
<context-param>  
    <param-name> javax.faces.PROJECT_STAGE </param-name>  
    <param-value> Development </param-value>  
</context-param>
```
- Essa propriedade de configuração, permite que o JSF emita mensagens de erros mais completas no cliente, facilitando o desenvolvimento.

- Os valores possíveis são Development, UnitTest, SystemTest, Production, Extension.

```
#{facesContext.application.projectStage}
```

```
#{initParam['javax.faces.PROJECT_STAGE']}
```

- //Verificando através do enum de ProjectStage

```
FacesContext facesContext = FacesContext.getCurrentInstance();
```

```
if (facesContext.isProjectStage(ProjectStage.Development)) {
```

```
    // executa determinada ação
```

```
}
```

- Provê meios mais simples de criar um componente customizado, sem a necessidade de criar tlds ou estender Renders.
- Simplesmente utilizando tag é possível criar um componente.

```
<html... xmlns:cc="http://java.sun.com/jsf/composite/">
```

```
<cc:interface>
```

```
<cc:attribute name="title"/>
```

```
<cc:attribute name="description" required="true"/>
```

```
<cc:attribute name="actionMethod" method-signature="java.lang.String  
adicionar()" />
```

- </cc:interface>

..

Composite Components

```
<cc:implementation>
  <h:form>
    #{cc.attrs.title} - #{cc.attrs.description}
    <h:commandButton action="#"#{cc.attrs.actionMethod}"
value="Confirmar"/>
  </h:form>
</cc:implementation>
</html>
```


Composite Components






- Utilizando um componente criado:

```
<html... xmlns:mo="http://java.sun.com/jsf/composite/moviecomp">
```

...

```
<mo:moviebox title="#{movie.title}" actionMethod=""/>
```

```
</html>
```

- ▼  resources
 - ▶  css
 - ▶  images
 - ▼  moviecomp
 - ▶  moviebox.xhtml

- Integração com as especificações JSR 299 (CDI) e JSR 330 (DI)
- JSR 299: Provê uma arquitetura com escopes bem definidos. Também traz transiociionalidade a camada web.
- JSR 330: Provê conjunto de anotações para realizar injeção de dependência.
- É possível utilizar diretamente na view qualquer um EJB que utiliza CDI.

@Stateless

@LocalBean

@Named("movieService")

```
public class MovieManagerSessionBean {  
    @PersistenceContext(unitName = "movie-persistence")  
    private EntityManager entityManager;  
    ....  
}
```

```
<h:dataTable value="#{movieService.allMovies}" var="movie"  
    styleClass="tabela" headerClass="header" rowClasses="odd,even">
```

JSF 2.2

- Nova especificação – JSR 344
- Será lançado antes do Java EE 7
- Proposed Final Draft Published = Março de 2012

JSF 2.2 – File Upload

- File Upload nativo.
- Suporte a AJAX.
- Compatível com Portlets
- Será criado um novo componente `<h:inputFile />`
- Possuirá atributos que permitirá o desenvolvedor habilitar ou não o uso de ajax, javascript, portlet e entre outros.

JSF 2.2 – viewAction Component

- Com o JSF 2.2 existe um novo componente chamado de viewAction que proporciona uma maneira mais fácil e rápida para fazer determinadas validações no lado do servidor
- Similar ao componente `<s:viewAction>` do Seam 3
- `<f:viewAction>` é declarado como child de `<f:metadata>`

```
<f:metadata>
```

```
    <f:viewParam name="id" value="#{produto.id}"/>
```

```
    <f:viewAction action="#{produto.verificaProduto}"/>
```

```
</f:metadata>
```

JSF 2.2 – viewAction Component

```
public String verificaProduto() {  
    boolean isExists = verificaProduto(produto.id);  
    if(!isExists)  
        facesContext.addMessage(null, new FacesMessage("Produto  
        não encontrado"));  
    return null;  
}
```

- Links bookmarkable mais eficientes. Maior controle dos parâmetros GET

JSF 2.2 – viewAction Component

- A especificação do JSF 2 exige pelo menos um view parameter dentro de um view metadata para que ele seja executado, porém, o JSF 2.2 não possui esse pré-requisito
- Pode ser usado em qualquer fase do ciclo de vida do JSF

```
<f:viewAction action="#{catalog.checkItem}"
phase="UPDATE_MODEL_VALUES"/>
```
- Especificamos a fase do ciclo de vida do JSF através da constante da classe `javax.faces.event.PhaseID`

JSF 2.2 – viewAction e Navegação Implícita

- Compatível com regras de navegação

```
<navigation-rule>
```

```
    <from-view-id>/index.xhtml</from-view-id>
```

```
    <navigation-case>
```

```
        <from-action> #{produto.verificaProduto}
```

```
        </from-action>
```

```
        <if>#{!produto.isInvalido} </if>
```

```
        <to-view-id>/listaProdutos.xhtml</to-view-id>
```

```
        <redirect/>
```

```
    </navigation-case>
```

```
</navigation-rule>
```

JSF 2.2 – DI em todos os artefatos JSF

- Até o JSF 2.1 somente era possível fazer DI para EJBs (`@EJB`), CDI (`@Inject`) e managed beans do JSF
- Mas e os Converters? Validators? Como eu faço DI de um EJB neles?
- JSF 2.0 e 2.1 = lookup
- JSF 2.2 = DI

JSF 2.2 – DI em todos os artefatos JSF



```
@FacesConverter(value="br.com.fourlinux.videostore.converters.Movie")
```

```
public class MovieConverter implements Converter {
```

```
@Override
```

```
public Object getAsObject(FacesContext context,  
UIComponent component, String value) {
```

```
MovieManagerSessionBean movies;
```

```
try {
```

```
Context jndiContext = new InitialContext();
```

```
movies = (MovieManagerSessionBean) jndiContext  
.lookup("java:module/MovieManagerSessionBean");
```

```
...
```

JSF 2.2 – DI em todos os artefatos JSF



```
@FacesConverter(value="br.com.fourlinux.videostore.converters.Movie")

public class MovieConverter implements Converter {

    @Override

    public Object getAsObject(FacesContext context,
        UIComponent component, String value) {

        @EJB MovieManagerSessionBean movies;

        ...
    }
}
```

JSF 2.2 – Vários outros recursos



- Gerenciar requisições ajax em um fila (queue control)
- Suporte a HTML 5
- Maior integração com portlets

JSF 2 – Frameworks

- RichFaces 4 - <http://www.jboss.org/richfaces>
- PrimeFaces 2 - <http://www.primefaces.org/>
- IceFaces 2 - <http://www.icefaces.org/>
- MyFaces Trinidad 2.0 - <http://myfaces.apache.org/trinidad>

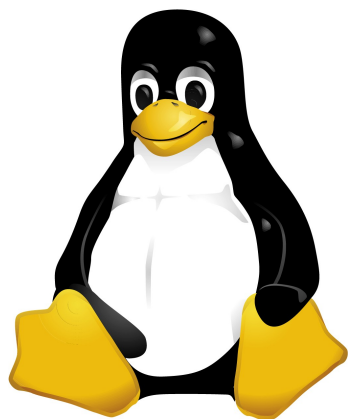
Referências

- <http://www.jcp.org/en/jsr/detail?id=314>
- <http://docs.jboss.org/seam/3/faces/latest/reference/en-US/html/components.html#viewaction>
- <http://java.net/jira/secure/IssueNavigator.jspa?mode=hide&requestId=10268>
- <http://jaserverfaces-spec-public.java.net/>

Código fonte

- <https://github.com/gabriel-ozeas/javaone2011>
- Para executar é necessário o maven
- `mvn clean package cargo:run`
- `localhost:8080` cai direto na aplicação!

Obrigado



Gustavo Lira
Gabriel Ozeas

gustavo@4linux.com.br
gabriel.ozeas@4linux.com.br
www.4linux.com.br
www.hackerteen.com
twitter.com/4LinuxBR

Tel: 55-11-2125-4747