



Curso JSF

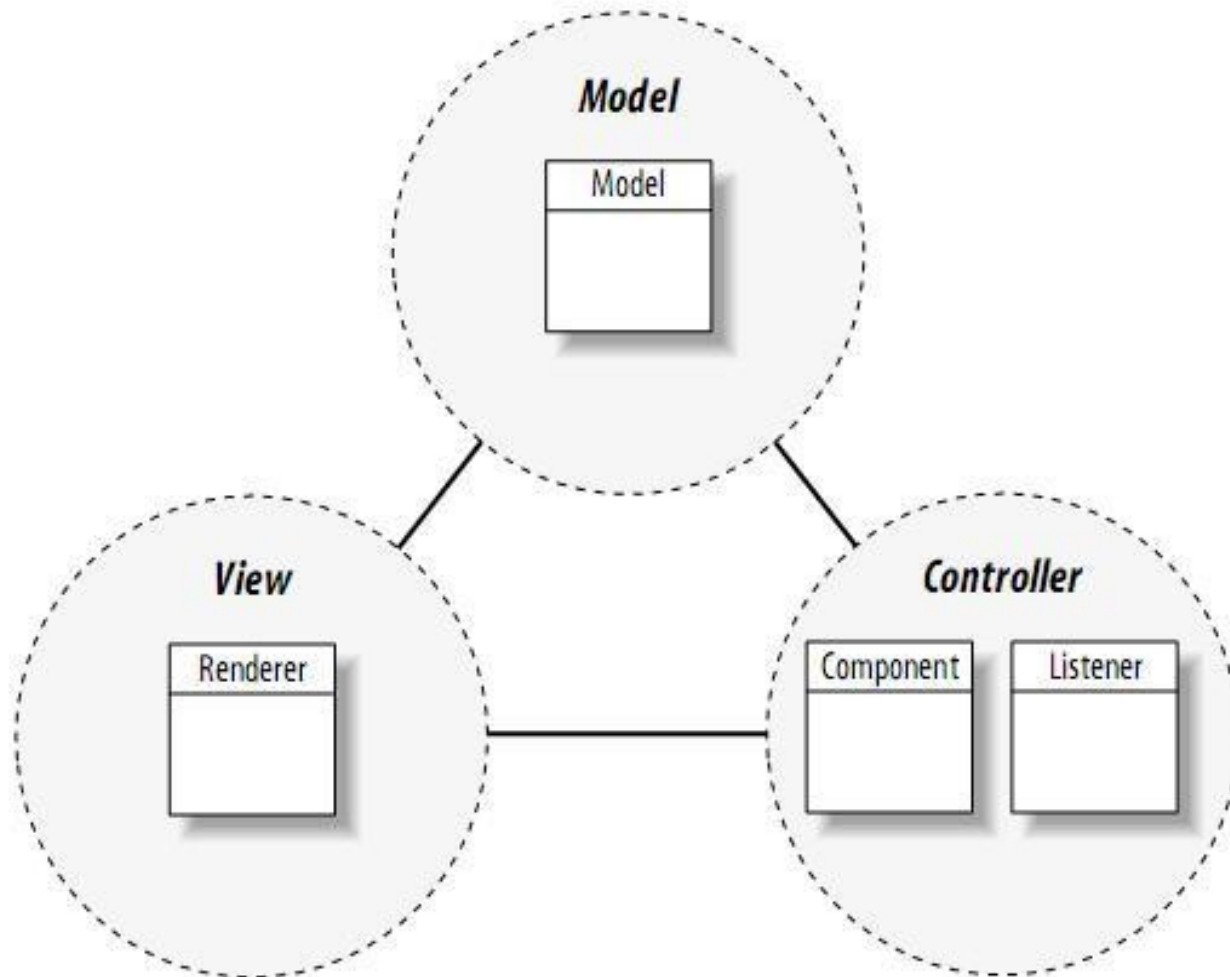
Fábio Henrique Barros

- Java Server Faces
 - Introdução
 - Gerenciando a Navegação
 - Managed Beans
 - JSF Expression Language
 - Java Server Faces Standard Tags
 - Modelo de Componentes
 - Conversão e Validação
 - Tratando Eventos
 - Ciclo de Vida de uma Requisição
 - Suporte ao método GET
 - Ajax e JSF

- O que é o Java Server Faces (3ª Geração)
 - Framework para simplificar o desenvolvimento de aplicações WEB
 - Baseado em Componentes
 - Conjunto de componentes GUI para WEB
 - Modelo orientado a eventos
 - Ciclo de vida de requisição bem definido.
 - O desenvolvedor não precisa se preocupar com detalhes sobre HTTP
 - Não é limitado a HTML
 - Container IoC (provê injeção de dependências)

- O que é o Java Server Faces (cont.)
 - Vantagens
 - Componentes GUI personalizados
 - POJOs (Plain Old Java Object):
 - Objeto sem herança nem implementa interfaces.
 - Tratamento de Eventos
 - Conversão e Validação
 - Suporte a outras tecnologias de transporte e apresentação
 - Managed Beans
 - Expression Language
 - Configuração mais simples

- Design MVC



- Primeira Aplicação
 - Bibliotecas (Java.net RI)
 - jsf-api.jar
 - jsf-impl.jar
 - Configurações
 - web.xml
 - faces-config.xml
 - Páginas
 - jsp, xhtml, html, js, css, etc...

■ Exemplo: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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-app_4_0.xsd"
  id="WebApp_ID" version="4.0">
  <display-name>CursoJsf</display-name>
  <module-name>cursojsf</module-name>
  <welcome-file-list>
    <welcome-file>index.xhtml</welcome-file>
  </welcome-file-list>
  <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>
</web-app>
```

■ Exemplo: faces-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<faces-config 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/javaee  
        http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"  
    version="2.0">
```

...

```
</faces-config>
```


■ Exemplo: index.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">

    <h:head>
        <title>
            <h:outputText value="Pagina Inicial" />
        </title>
    </h:head>

    <h:body>
        <h:outputText value="Pagina Inicial" />
    </h:body>

</html>
```

- Fluxo da requisição:
 - GET para index.xhtml
 - Faces Servlet intercepta
 - Localiza a página
 - Interpreta
 - Responde o HTML gerado

■ Gerenciando a Navegação

- Estática – Configurado na própria página
- Dinâmica – Saída de uma Action
- Definida usando Regras de Navegação

- Configurado no faces-config.xml

```
<navigation-rule>
  <from-view-id>/login.xhtml</from-view-id><!--opcional-->
  <navigation-case>
    <from-action>#{loginBean.login}</from-action><!--opcional-->
    <from-outcome>sucesso</from-outcome>
    <to-view-id>/index.xhtml</to-view-id>
    <redirect /><!--opcional-->
  </navigation-case>
</navigation-rule>
```

- Navegação implícita

- Se nenhum caso de navegação correspondente for encontrado, o NavigationHandler verifica se o resultado da ação corresponde a uma página e executa navegação para a mesma.

■ Exemplo de configuração: faces-config.xml

```
...  
<!-- Navigation -->  
<navigation-rule>  
    <navigation-case>  
        <from-outcome>index</from-outcome>  
        <to-view-id>/index.xhtml</to-view-id>  
        <redirect />  
    </navigation-case>  
  
    <navigation-case>  
        <from-outcome>ajuda</from-outcome>  
        <to-view-id>/ajuda.xhtml</to-view-id>  
        <redirect />  
    </navigation-case>  
</navigation-rule>  
...
```

■ Exemplo: index.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">

    <h:head>
        <title>
            <h:outputText value="Pagina Inicial" />
        </title>
    </h:head>

    <h:body>
        <h:form>
            <h:commandLink action="ajuda?faces-redirect=true"
                value="Pagina de ajuda" />
        </h:form>
    </h:body>

</html>
```

■ Exemplo: ajuda.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">

    <h:head>
        <title>
            <h:outputText value="Pagina de Ajuda" />
        </title>
    </h:head>

    <h:body>
        <h:form>
            <h:outputText value="Texto da ajuda!" />
            <br />
            <h:commandButton action="index" value="Voltar" />
        </h:form>
    </h:body>

</html>
```

■ Exercício:

- Crie uma página de sobre, onde esta deverá conter as informações do sistema e do autor
- Crie uma navegação da página de ajuda para a página de sobre
- Crie uma navegação da página de sobre para a página de ajuda

- Message Bundles
 - Centralizar mensagens em um único lugar
 - Arquivo .properties do tipo chave=valor
 - Suporte a mensagens com parâmetros
 - Suporte a internacionalização

■ Configuração:

■ SystemMessages.properties

```
#Page: index
page.index.title=Página Inicial
page.index.link.ajuda=Visualizar ajuda!
#Page: ajuda
page.ajuda.title=Página de Ajuda
page.ajuda.texto=Texto da ajuda!
```

■ faces-config.xml

```
<faces-config>
  <!-- Configurações gerais do JSF -->
  <application>
    <resource-bundle>
      <base-name>SystemMessages</base-name>
      <var>msg</var>
    </resource-bundle>
    <locale-config>
      <default-locale>pt_BR</default-locale>
      <supported-locale>en</supported-locale>
    </locale-config>
  </application>
</faces-config>
```

■ Exemplo: index.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">

    <h:head>
        <title>
            <h:outputText value="#{msg['page.index.title']}" />
        </title>
    </h:head>

    <h:body>
        <h:form>
            <h:commandLink action="/ajuda?faces-redirect=true"
                value="#{msg['page.ajuda.title']}" />
        </h:form>
    </h:body>

</html>
```

■ Exemplo: ajuda.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
<h:head>
    <title>
        <h:outputText value="#{msg['page.ajuda.title']}" />
    </title>
</h:head>
<h:body>
    <h:form>
        <h:outputText value="#{msg['page.ajuda.texto']}" />
        <br />
        <h:commandLink action="sobre?faces-redirect=true"
            value="#{msg['page.sobre.title']}" />
        <br />
        <h:commandButton action="index?faces-redirect=true"
            value="#{msg['commons.voltar']}" />
    </h:form>
</h:body>
</html>
```

■ Exemplo: sobre.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">

<h:head>
    <title>
        <h:outputText value="#{msg['page.sobre.title']}" />
    </title>
</h:head>

<h:body>
    <h:form>
        <h:outputText value="#{msg['page.sobre.autor']}" />
        <br />
        <h:commandButton action="ajuda?faces-redirect=true"
            value="#{msg['commons.voltar']}" />
    </h:form>
</h:body>

</html>
```

■ Managed Beans

- Usados para separação entre as lógicas de negócio e de apresentação
- O que são Beans?
 - Definição de JavaBean : “a reusable software component that can be manipulated in a builder tool”
 - Construtor Vazio
 - Propriedades privadas, acessadas por métodos get e set (getXyz, setXyz)
 - Não é necessário herdar de nenhuma classe em especial (POJO)
 - No JSF também são chamados de backing beans
- Gerenciados pelo Framework

- Managed Beans (cont.)
 - Definidos no faces-config.xml ou annotations
 - Nome, classe, propriedades e escopo
 - Propriedades para os dados do Form
 - Métodos Action (Controller)
 - Propriedades para valores de saída
 - Associados a páginas WEB através da EL
 - Instanciados e inicializados pelo Container

- Managed Beans (Escopos do JSF 2.0)
 - @NoneScoped
 - O bean não possui escopo. Utiliza o escopo dos beans que o referencia.
 - @RequestScoped
 - Escopo padrão.
 - Criado a cada requisição.
 - @ViewScoped
 - Instância do bean é criada e mantida enquanto o usuário estiver na página (por exemplo, com os manipuladores de eventos ou Ajax).
 - Deve implementar Serializable

- Managed Beans (Escopos do JSF 2.0)
 - @SessionScoped
 - Criado um por sessão de cada usuário.
 - @ApplicationScoped
 - Criado um para toda a aplicação.
 - @CustomScope
 - O bean é armazenado em um Mapeamento (Map), e programador pode controlar ciclo de vida.

■ Exemplo: GuessNumberBean.java

```
@SessionScoped
@ManagedBean(name = "guessBean")
public class GuessNumberBean {
    /** Numero a ser adivinado. */
    private Integer numero;
    /** Palpite do usuário. */
    private Integer palpite;
    /** Tentativas. */
    private Integer tentativas;
    /** Mensagem de erro. */
    private String mensagem;
    /* Gerar get e set. */
    public String init() {
        numero = (int) (1 + (Math.random() * 100));
        palpite = null;
        tentativas = 0;
        mensagem = "page.guess.label.branco";
        return "guess";
    }
    public String guess() {
        if (palpite.equals(numero)) {
            mensagem = "page.guess.acerto";
        } else if (numero.compareTo(palpite) < 0) {
            mensagem = "page.guess.menor";
        } else {
            mensagem = "page.guess.maior";
        }
        tentativas++;
        return "guess";
    }
}
```

- Exemplo: faces-config.xml
 - Opcional no JSF 2.0

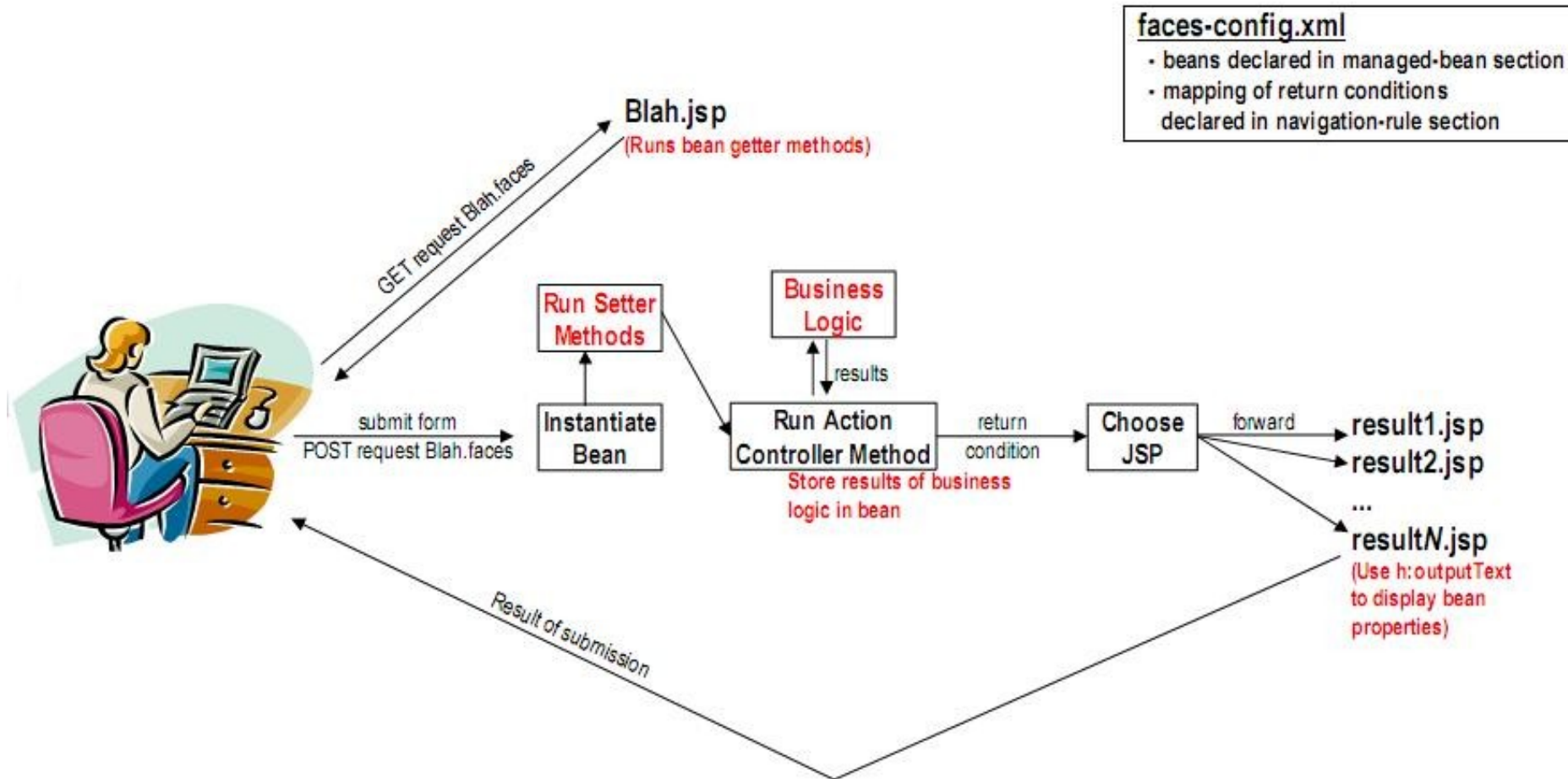
```
<faces-config>
...
<!-- Managed Beans -->
<managed-bean>
  <managed-bean-name>guessBean</managed-bean-name>
  <managed-bean-class>
    br.com.cursojsf.managedbean.GuessNumberBean
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>tentativas</property-name>
    <value>0</value>
  </managed-property>
</managed-bean>
...
<navigation-rule>
  <navigation-case>
    <from-outcome>guess</from-outcome>
    <to-view-id>/guess.xhtml</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
...
</faces-config>
```

■ Exemplo: guess.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
<h:head>
    <title><h:outputText value="#{msg['page.guess.title']}" /></title>
</h:head>
<h:body>
    <h:form id="guessForm">
        <h:commandLink id="link" action="#{guessBean.init}"
            value="#{msg['page.guess.label.iniciar']}" /> <br/><br/>
        <h:outputLabel for="palpite"
            value="#{msg['page.guess.label.numero']}: " />
        <h:inputText id="palpite" value="#{guessBean.palpite}" />
        <h:commandButton value="#{msg['page.guess.label.enviar']}"
            action="#{guessBean.guess}" /> <br /><br />
        <h:outputFormat id="outtext" value="#{msg[guessBean.mensagem]}"
            styleClass="error">
            <f:param value="#{guessBean.tentativas}" />
        </h:outputFormat>
        <hr />
        <h:commandButton action="index" value="#{msg['commons.voltar']}" />
    </h:form>
</h:body>
</html>
```

JSF - Managed Beans

■ Ciclo de uma requisição:



■ JSF Expression Language

- Bastante semelhante a EL do JSP
 - Forma de uso: `#{blah}`
- Usado nas páginas, faces-config.xml e beans
- Fácil acesso as propriedades de um bean
 - `#{loginBean.email}`, `#{loginBean.senha}`,
`#{usuarioBean.usuario.nome}`,
`#{usuarioBean.usuario.endereco.logradouro}`
- Acesso a objetos implícitos (session, request, application)
- Acesso a propriedades e métodos dos beans
- Acesso simples a elementos de uma coleção
- Suporte a operadores (or, and, eq, lt, gt, +, -, etc)

■ JSF Standard Tags

- Disponibilizadas pelo JSF
- Componentes Base para desenvolvimento
- Três bibliotecas: html, core e ui (43 tags no total)
- São taglibs

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

- Biblioteca core
 - Serve como suporte para html (e outras)
 - Toda página de ter um `<f:view>`
 - Raiz da árvore de componentes
 - A partir da versão 2.0 não é mais necessário
 - Tag `<f:loadBundle>` carrega um “.properties”
 - A partir da versão 1.2 não é mais necessário
 - A maioria das tags representam objetos a serem adicionados a componentes
 - Atributos, Listeners, Converters, Validators, Facets, Parametros, Select Items

■ Biblioteca Html

■ Biblioteca de componentes Html

- Componentes para entrada e saída de dados
- Outros: forms, mensagens, componentes de layout

■ Tags da lib html

- `form, inputText, inputTextArea, inputSecret, inputHidden, outputLabel, outputLink, outputFormat, outputText, commandButton, commandLink, message, messages, graphicImage, selectOneListBox, selectOneMenu, selectOneRadio, selectBooleanCheckbox, SelectManyCheckbox, SelectManyListbox, SelectManyMenu, panelGrid, panelGroup, dataTable, column...`

- Biblioteca html

- Categorias

- Input (`input...`)
 - Output (`output...`)
 - Comandos ou actions (`commandButton`, `commandLink`)
 - Seleção (`checkbox`, `listbox`, `menu`, `radio`)
 - Layout (`panelGrid`, `panelGroup`)
 - Data Table (`dataTable`)
 - Erros e Mensagens (`message`, `messages`)

- Biblioteca html
 - Atributos Comuns a todos
 - HTML 4.0 (accesskey, lang, dir, etc) e eventos DHTML (onclick, onmouseover, onchange, etc)
 - **id** – Identificador para o componente
 - **binding** – Usado para referenciar componente em um backing bean
 - **rendered** – Booleano indicando se o componente será mostrado
 - **styleClass** – Nome da classe CSS

- Biblioteca html

- Outros Atributos

- **value** – Valor do componente (I, O, C)
 - **valueChangeListener** – Método que responde a uma mudança no valor do componente (I)
 - **converter** – Conversor associado ao componente (I, O)
 - **validator** – Validador associado ao componente (I)
 - **required** – Booleano indicando se o campo é obrigatório (I)

■ DataTable

- Componente para mostrar dados de forma tabular
- Renderiza a tag <table> do HTML
- Não é necessário Loops. É só usar
- Definição de Cabeçalho e Rodapé
- É configurável por CSS
 - headerClass – classe css para o cabeçalho
 - footerClass – classe css para o rodapé
 - rowClasses – classe para as linhas (lista separada por vírgula)

■ Exemplo: usuarios.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">

<h:head>
    <title>
        <h:outputText value="#{msg['page.usuarios.title']}" />
    </title>
</h:head>

<h:body>
<h:form>
    <h:dataTable value="#{usuarioBean.listaUsuarios}" var="usuario"
        border="1" cellpadding="2" cellspacing="0">
        <h:column>
            <f:facet name="header">
                <h:outputText value="#{msg['page.usuarios.label.cpf']}" />
            </f:facet>
            <h:outputText value="#{usuario.cpf}" />
        </h:column>
    ...

```

■ Exemplo: usuarios.xhtml (cont.)

```
<h:column>
  <f:facet name="header">
    <h:outputText
      value="#{msg['page.usuarios.label.nomeReduzido']}" />
  </f:facet>
  <h:outputText value="#{usuario.nomeReduzido}" />
</h:column>
<h:column>
  <f:facet name="header">
    <h:outputText value="#{msg['page.usuarios.label.email']}" />
  </f:facet>
  <h:outputLink value="mailto:#{usuario.email}">
    <h:outputText value="#{usuario.email}" />
  </h:outputLink>
</h:column>
<h:column>
  <f:facet name="header">
    <h:outputText value="#{msg['page.usuarios.label.dataNascimento']}" />
  </f:facet>
  <h:outputText value="#{usuario.dataNascimento}" />
</h:column>
```

...

■ Exemplo: usuarios.xhtml (cont.)

```
</h:dataTable>
<hr/>
<h:commandButton action="index" value="#{msg['commons.voltar']}" />
</h:form>
</h:body>
</html>
```

■ Exemplo: UsuarioBean.java

@ManagedBean

@RequestScoped

```
public class UsuarioBean {  
    /** Referencia para a camada de regras de negocio */  
    @ManagedProperty("#{usuarioBusiness}")  
    private UsuarioBusiness usuarioBusiness;  
    /** Usuario a ser usado no form. */  
    private Usuario usuario = new Usuario();  
    public UsuarioBusiness getUsuarioBusiness() {  
        return usuarioBusiness;  
    }  
    public void setUsuarioBusiness(UsuarioBusiness usuarioBusiness) {  
        this.usuarioBusiness = usuarioBusiness;  
    }  
    public Usuario getUsuario() {  
        return usuario;  
    }  
    public void setUsuario(Usuario usuario) {  
        this.usuario = usuario;  
    }  
    public List<Usuario> getListaUsuarios() {  
        return usuarioBusiness.selecionarTodos();  
    }  
}
```


- Exercício:
 - Construa a consulta de usuários
 - Construa o cadastro de usuários
 - Página de consulta deve ter:
 - Um link para edição
 - Botão para adicionar um novo usuário
 - Botão para excluir usuário
 - Página de edição deve ter:
 - Form para alterar/incluir usuário
 - Veja os exemplos a seguir

■ Exemplo: usuarios.xhtml

...

```
<h:commandLink action="usuariosEditar">
  <h:outputText value="#{usuario.cpf}" />
  <f:setPropertyActionListener target="#{usuarioBean.usuario}"
    value="#{usuario}" />
</h:commandLink>
```

```
<h:column>
  <f:facet name="header">
    <h:outputText value="#{msg['commons.excluir']}" />
  </f:facet>
  <h:commandButton action="#{usuarioBean.excluir}"
    value="#{msg['commons.excluir']}">
    <f:setPropertyActionListener target="#{usuarioBean.usuario}"
      value="#{usuario}" />
  </h:commandButton>
</h:column>
```

...

```
<h:commandButton action="#{usuarioBean.novo}" value="#{msg['commons.novo']}" />
```

■ Exemplo: usuariosEditor.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
<h:head>
    <title>
        <h:outputText value="#{msg['page.usuariosEditor.title']}" />
    </title>
</h:head>
<h:body>
    <h:form id="usuarioForm">
        <h:inputHidden value="#{usuarioBean.usuario.id}" />
        <h:panelGrid columns="2">
            <h:outputLabel for="idInput" value="#{msg['page.usuarios.label.id']}: " />
            <h:inputText id="idInput" value="#{usuarioBean.usuario.id}"
                disabled="true" />

            <h:outputLabel for="cpfInput" value="#{msg['page.usuarios.label.cpf']}: " />
            <h:inputText id="cpfInput" value="#{usuarioBean.usuario.cpf}" />

            <h:outputLabel for="nomeReduzidoInput"
                value="#{msg['page.usuarios.label.nomeReduzido']}: " />
            <h:inputText id="nomeReduzidoInput"
                value="#{usuarioBean.usuario.nomeReduzido}" />
        </h:panelGrid>
    </h:form>
</h:body>
</html>
```

■ Exemplo: usuariosEditor.xhtml

```
<h:outputLabel for="nomeCompletoInput"
  value="#{msg['page.usuarios.label.nomeCompleto']}:" />
<h:inputTextarea id="nomeCompletoInput"
  value="#{usuarioBean.usuario.nomeCompleto}" rows="5" cols="20"/>

<h:outputLabel for="emailInput"
  value="#{msg['page.usuarios.label.email']}:" />
<h:inputText id="emailInput" value="#{usuarioBean.usuario.email}" />

<h:outputLabel for="senhaInput"
  value="#{msg['page.usuarios.label.senha']}:" />
<h:inputText id="senhaInput"
  value="#{usuarioBean.usuario.senha}" />
</h:panelGrid>
<hr/>
<h:commandButton action="#{usuarioBean.salvar}"
  value="#{msg['commons.salvar']}:" />
<h:commandButton action="usuarios"
  value="#{msg['commons.voltar']}:" />
</h:form>
</h:body>
</html>
```

■ Exemplo: faces-config.xml (opcional)

```
<navigation-rule>
  <navigation-case>
    <from-outcome>usuariosEditar</from-outcome>
    <to-view-id>/usuariosEditar.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

■ Exemplo: UsuarioBean.java

```
public String novo() {
    usuario = new Usuario();
    return "usuariosEditar";
}

public String salvar() {
    usuarioBusiness.salvarUsuario(usuario);
    return "usuarios";
}

public String excluir() {
    usuarioBusiness.excluirUsuario(usuario);
    return "usuarios";
}
```

- Recursos de implementação
 - XHTML: `<f:setPropertyActionListener>`
 - Carrega propriedade de um Managed Bean
 - Destino: `target="#{usuarioBean.usuario}"`
 - Origem: `value="#{usuario}"`
 - Faces-config.xml: `<redirect />`
 - Envia comando http para o browser efetuar nova requisição para a página desejada
 - Não deve ser usada em conjunto com beans que estão em escopo de request
 - Como o browser faz nova requisição, apenas os objetos que estão na sessão estarão visíveis.

■ Modelo de Componentes

- Modelo de componentes UI: Define o conjunto de classes que especifica o estado e o comportamento dos componentes UI.
- Modelo de Renderização: Define como renderizar (“mostrar”) os componentes de diferentes maneiras.
- Modelo de Eventos e Listeners: Define como tratar os eventos dos componentes.
- Modelo de conversão: Define como diferentes conversores são “plugados” aos componentes.
- Modelo de validação: Define como registrar validadores em um componente.

■ FacesContext

- Contém todas as informações sobre estado relacionadas ao processamento de uma única requisição.
- É passado para, e potencialmente modificado por, cada fase do ciclo de vida da requisição.
- Acessado através de el `#{facesContext}` ou nos Beans através do método `FacesContext.getCurrentInstance()`

■ Exemplo: FacesContext

```
public String autenticar() {  
    try {  
        usuarioLogado =  
            usuarioBusiness.autenticarUsuario(  
                usuario.getCpf(), usuario.getSenha());  
        return "principal";  
    } catch (UsuarioInvalidoException e) {  
        FacesMessage message = new FacesMessage();  
        message.setSeverity(FacesMessage.SEVERITY_ERROR);  
        message.setDetail("Usuário ou senha inválidos!");  
        FacesContext.getCurrentInstance()  
            .addMessage("loginForm", message);  
        return null;  
    }  
}
```

- Conversão e Validação – Converters
 - Em aplicações WEB os dados enviados do cliente são sempre Strings
 - Java é uma linguagem tipada
 - Precisamos converter os dados enviados
 - JSF provê Conversores para todos os tipos básicos do Java (Date, Integer, Double, Long)
 - Convertem o valor dos componentes de e para string
 - Aplicam formatação ou internacionalização
 - Podemos definir novos conversores e associar aos componentes

■ Exemplo: converters

```
...  
<h:form>  
    ...  
    <h:outputText value="#{msg['page.usuarios.cpf']}" />  
    <h:inputText id="ganhosUsuario" value="#{usuarioBean.usuario.ganhos}">  
        <f:convertNumber type="currency"/>  
    </h:inputText>  
    <h:message for="ganhosUsuario" />  
  
    <h:outputText value="#{msg['page.usuarios.cpf']}" />  
    <h:inputText id="cpfUsuario" value="#{usuarioBean.usuario.cpf}">  
        converter="converters.CpfConverter" />  
    <h:message for="cpfUsuario" />  
  
    <h:outputText value="#{msg['page.usuarios.dataNascimento']}" />  
    <h:inputText id="dataNascimentoUsuario">  
        value="#{usuarioBean.usuario.dataNascimento}">  
        <f:convertDateTime pattern="dd/MM/yyyy"/>  
    </h:inputText>  
    <h:message for="dataNascimentoUsuario" />  
    ...  
</h:form>  
...
```

- Conversão e Validação – Converters
 - É possível definir Converters
 - Implementar Interface `javax.faces.convert.Converter`
 - Métodos `getAsString` e `getAsObject`
 - Registrar o converter no `faces-config.xml` ou annotations.
 - Defini-se um id para o converter
 - Associa o converter a uma classe . Todas as instâncias dessa classe utilizaram esse converter (`<converter-for-class>`)
 - Associar ao componente desejado

■ Exemplo: CpfConverter.java

```
@FacesConverter("converters.CpfConverter")
public class CpfConverter implements Converter {
    public Object getAsObject(FacesContext context, UIComponent component,
        String value) throws ConverterException {
        if (value != null && !value.equals("")) {
            String cpf = value.replaceAll("\\.", "").replaceAll("\\-", "");
            try {
                // Testa se somente existem numeros.
                Long.valueOf(cpf);
                return cpf;
            } catch (NumberFormatException e) {
                FacesMessage message = new FacesMessage(FacesMessage.SEVERITY_ERROR,
                    "Erro de conversão", "O valor informado não é um número de CPF!");
                throw new ConverterException(message);
            }
        }
        return value;
    }
    public String getAsString(FacesContext context, UIComponent component,
        Object value) throws ConverterException {
        String cpf = (value == null? null: value.toString());
        if (cpf != null && !cpf.equals("")) {
            cpf = cpf.substring(0, 3) + "." + cpf.substring(3, 6) + "."
                + cpf.substring(6, 9) + "-" + cpf.substring(9);
        }
        return cpf;
    }
}
```

■ Exemplo: faces-config.xml (opcional)

```
<faces-config>
    ...
    <converter>
        <converter-id>converters.CpfConverter</converter-id>
        <converter-class>
            br.com.cursojsf.view.converters.CpfConverter
        </converter-class>
    </converter>
    ...
</faces-config>
```

■ Exemplo: usuariosEditar.xhtml

```
<h:inputText id="cpfInput" value="#{usuarioBean.usuario.cpf}"
    converter="converters.CpfConverter" />
```

OU

```
<h:inputText id="cpfInput" value="#{usuarioBean.usuario.cpf}">
    <f:converter converterId="converters.CpfConverter" />
</h:inputText>
```

- Conversão e Validação – Validators
 - Tarefa que praticamente toda aplicação precisa
 - Verifica se todos os campos obrigatórios foram preenchido corretamente
 - Mostrar página novamente se ocorrer algum problema
 - Para campos obrigatórios o JSF provê o atributo required
 - Provê também validação básicas (domínio e tamanho)
 - `<f:validateDoubleRange maximum="2" minimum="1"/>`
 - `<f:validateLongRange maximum="2" minimum="1"/>`
 - `<f:validateLength maximum="3" minimum="3"/>`
 - Podemos definir nosso próprio Validator
 - Por default não suporta validação do lado cliente

■ Exemplo: validators

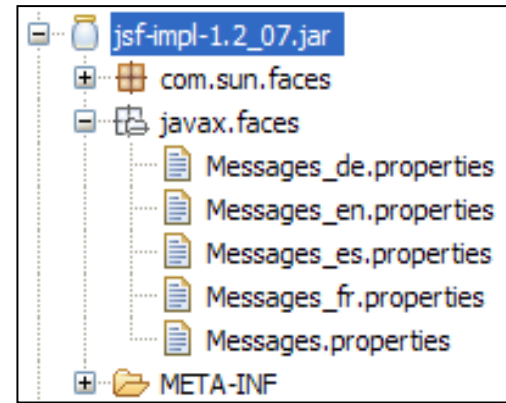
```
<h:outputText value="#{msg['page.usuarios.nome']}: " />
<h:inputText id="nomeUsuario" value="#{usuarioBean.usuario.nome}"
    required="true">
    <f:validateLength minimum="5"/>
</h:inputText>
<h:message for="nomeUsuario" errorClass="erro" />
<h:outputText value="#{msg['page.usuarios.dependentes']}: " />
<h:inputSecret id="numeroDependentesUsuario"
    value="#{usuarioBean.usuario.numeroDependentes}">
    <f:validateLongRange maximum="3"/>
</h:inputSecret>
<h:message for="senhaUsuario" errorClass="erro" />
<h:outputText value="#{msg['page.usuarios.senha']}: " />
<h:inputSecret id="senhaUsuario" value="#{usuarioBean.usuario.senha}"
    required="true">
    <f:validateLength minimum="6" maximum="10"/>
</h:inputSecret>
<h:message for="senhaUsuario" errorClass="erro" />
```

■ Exemplo: usuarioEditar.xhtml

```
<h:inputText id="cpfInput" value="#{usuarioBean.usuario.cpf}"
    required="true">
```


JSF - Conversão e Validação

- Dica: Como exibir as mensagens em Português Brasil?
 - Veja o conteúdo da biblioteca de implementação do JSF (jsf-impl-???.jar)
 - Abra o arquivo Messages.properties
 - Salve-o no pacote “javax.faces” no “src” do seu projeto com o nome de “FacesMessages_pt_BR.properties”
 - Edite o conteúdo do arquivo
- Será necessário adicionar no faces-config.xml:



```
<application>
```

```
...
```

```
<message-bundle>javax.faces.FacesMessages</message-bundle>
```

```
...
```

```
</application>
```

- Conversão e Validação – Validators
 - Duas maneiras de implementar validação
 - Implementar a interface `javax.faces.validator.Validator`
 - Criar a classe que será o Validador
 - Registrar no `faces-config.xml`
 - Associar aos componentes desejados
 - Usar um método do backing bean para validação
 - Método deve seguir a segunda assinatura:
`void validaCpf(FacesContext ctx, UIComponent comp, Object value) throws ValidatorException`
 - Associar ao componente usando el:

```
<h:inputText id="cpfUsuario"
value="#{usuarioBean.usuario.cpf}"
validator="#{usuarioBean.validaCpf}">
```

■ Exemplo: CpfValidator.java

```
@FacesValidator("validators.CpfValidator")
public class CpfValidator implements Validator {
    public void validate(FacesContext context, UIComponent component,
        Object value) throws ValidatorException {
        if (value != null) {
            String valor = value.toString();
            if (!ValidacaoHelper.validaCpf(valor)) {
                FacesMessage message = new FacesMessage();
                message.setSeverity(FacesMessage.SEVERITY_ERROR);
                message.setSummary("Erro de Validação");
                message.setDetail("CPF Inválido");
                throw new ValidatorException(message);
            }
        }
    }
}
```

■ Exemplo: faces-config.xml (opcional)

```
<faces-config>
    ...
    <validator>
        <validator-id>validators.CpfValidator</validator-id>
        <validator-class>
            br.com.cursojsf.view.validators.CpfValidator
        </validator-class>
    </validator>
    ...
</faces-config>
```

■ Exemplo: usuariosEditar.xhtml

```
<h:inputText id="cpfInput" value="#{usuarioBean.usuario.cpf}"
    required="true">
    <f:validator validatorId="validators.CpfValidator" />
</h:inputText>
...
<h:commandButton action="usuarios" value="#{msg['commons.voltar']}"
    immediate="true" />
```

- Atributo “immediate”
 - Presente em `<h:commandButton>` e `<h:commandLink>`
 - Informa ao faces que este deve pular a fase de `ApplyRequestValues` e `Validation`

- Duas variedades de **eventos** UI:
 - Eventos que disparam processamento back-end
 - Eventos que afetam apenas a forma da interface
- Usamos **listeners** para tratar esses eventos:
 - Action Listeners: envio completo do form
 - Disparados após preenchimento das propriedades e lógicas de validação
 - Disparados por `<h:commandButton>`, `<h:commandLink>`
 - Retorna uma String que afeta a navegação
 - Event Listeners: lidam com eventos da UI
 - Disparados antes do preenchimento das propriedades
 - Não afeta a navegação (continua na página atual)
 - Disparados por combo boxes, checkboxes, radio buttons, textfields e outros

■ Action Listeners

■ Método de um backing bean ou classe separada

- Método deve seguir a seguinte assinatura

```
public void nomeDoMetodo(ActionEvent event)
```

- Implementar interface

```
javax.faces.event.ActionListener
```

```
public interface ActionListener extends FacesListener {  
    void processAction(ActionEvent actionEvent)  
        throws AbortProcessingException;  
}
```

■ Associar listener a um componente

- Caso seja uma classe

```
<h:commandButton action="#{usuarioBean}"  
    value="Mostar/Esconder Ajuda" />
```

- Caso seja um método

```
<h:commandButton action="#{usuarioBean.mostraEsconderAjuda}"  
    value="Mostar/Esconder Ajuda" />
```

- ValueChangeListeners

- Método de um backing bean ou classe separada

- Método deve seguir a seguinte assinatura

- ```
public void nomeDoMetodo(ValueChangeEvent event)
```

- Implementar interface

- ```
javax.faces.event.ValueChangeListener
```

- ```
public interface ValueChangeListener extends FacesListener {
 void processValueChange(ValueChangeEvent event)
 throws AbortProcessingException;
}
```

- Associar listener a um componente

- Caso seja uma classe

- ```
<h:checkbox valueChangeListener="#{usuarioBean}"  
    value="#{usuarioBean.mostrarAjuda}"  
    onclick="this.form.submit()" />
```

- Caso seja um método

- ```
<h:selectOneMenu
 valueChangeListener="#{usuarioBean.mudarTipoRelatorio}"
 value="#{usuarioBean.tipoRelatorio}" />
```



## ■ Exemplo: guess.xhtml

```
<h:inputText id="palpite" value="#{guessBean.palpite}"
 valueChangeListener="#{guessBean.onChange}"
 onblur="this.form.submit()" ">
</h:inputText>
```

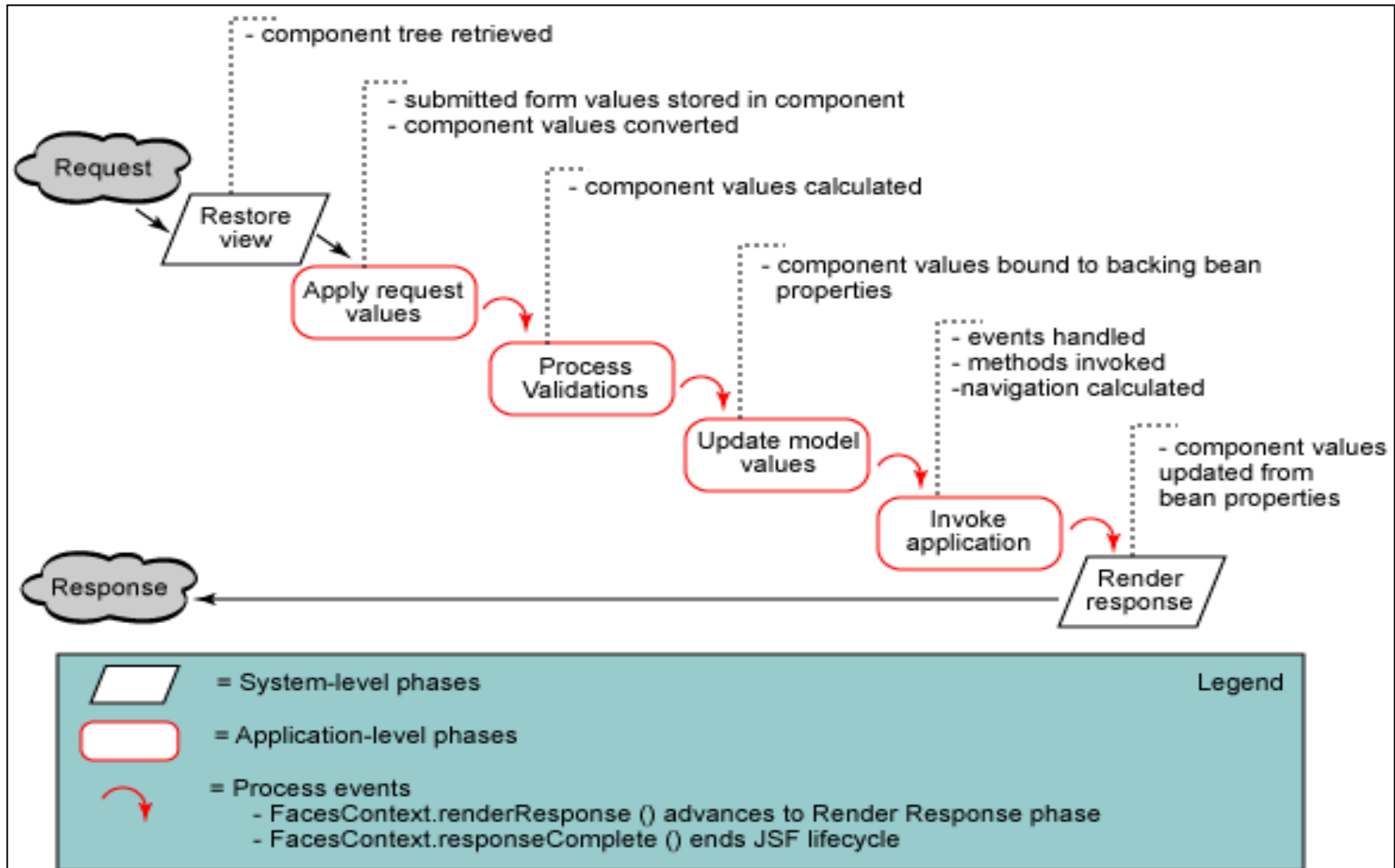
## ■ GuessBean.java

```
public void onChange(ValueChangeEvent event) {
 if (event.getNewValue() == null) {
 mensagem = "page.guess.digiteNumero";
 } else {
 mensagem = "page.guess.label.branco";
 }
}
```

- Existem seis fases no ciclo de vida de um requisição
  - 1. Restore view
  - 2. Apply request values
  - 3. Process validations
  - 4. Update model values
  - 5. Invoke application
  - 6. Render response

# JSF - Ciclo de Vida da Requisição

## ■ Modelo de requisição:



## ■ 1. Restore view

- O controlador examina a requisição que vem do FacesServlet e extrai o “view ID”, que é determinado pelo nome da página JSP/XHTML.
- View State Restore Method

- Pode ser configurado no web.xml

```
<context-param>
 <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
 <param-value>client</param-value>
</context-param>
```

- Dois valores possíveis:
  - client = salva o estado no browser (menor consumo de memória e maior tráfego de rede)
  - server = salva o estado no servidor (maior consumo de memória e menor tráfego de rede)

## ■ 2. Apply Request Values

- Cada componente deverá recuperar seu estado corrente.
- Os valores das propriedades dos componentes devem ser extraídas dos parâmetros do “request”, “cookies” ou do “header” da requisição.
- A atribuição do valor (value) é feita utilizando o “converter” definido para o componente.

## ■ 3. Process Validation

- Cada componente (propriedade “value”) será validado de acordo com as regras de validação definidas na aplicação.
- As regras de validação são definidas através dos “validators”.
- Cada valor é então validado e caso seja inválido, uma mensagem de erro é adicionada no FacesContext e o componente é marcado como inválido.
- Se um componente for marcado como inválido, o JSF pula para a fase “Render Response”.
- Caso nenhum componente esteja inválido o JSF passa para a próxima fase.

## ■ 4. Update Model Values

- Sua principal atividade é atribuir os valores informados pelo usuário no formulário, para as respectivas propriedades associadas aos BackBeans (também conhecidos como Managed Beans).
- Pode haver erro na atribuição, fazendo com que o JSF dispare um erro de tempo de execução

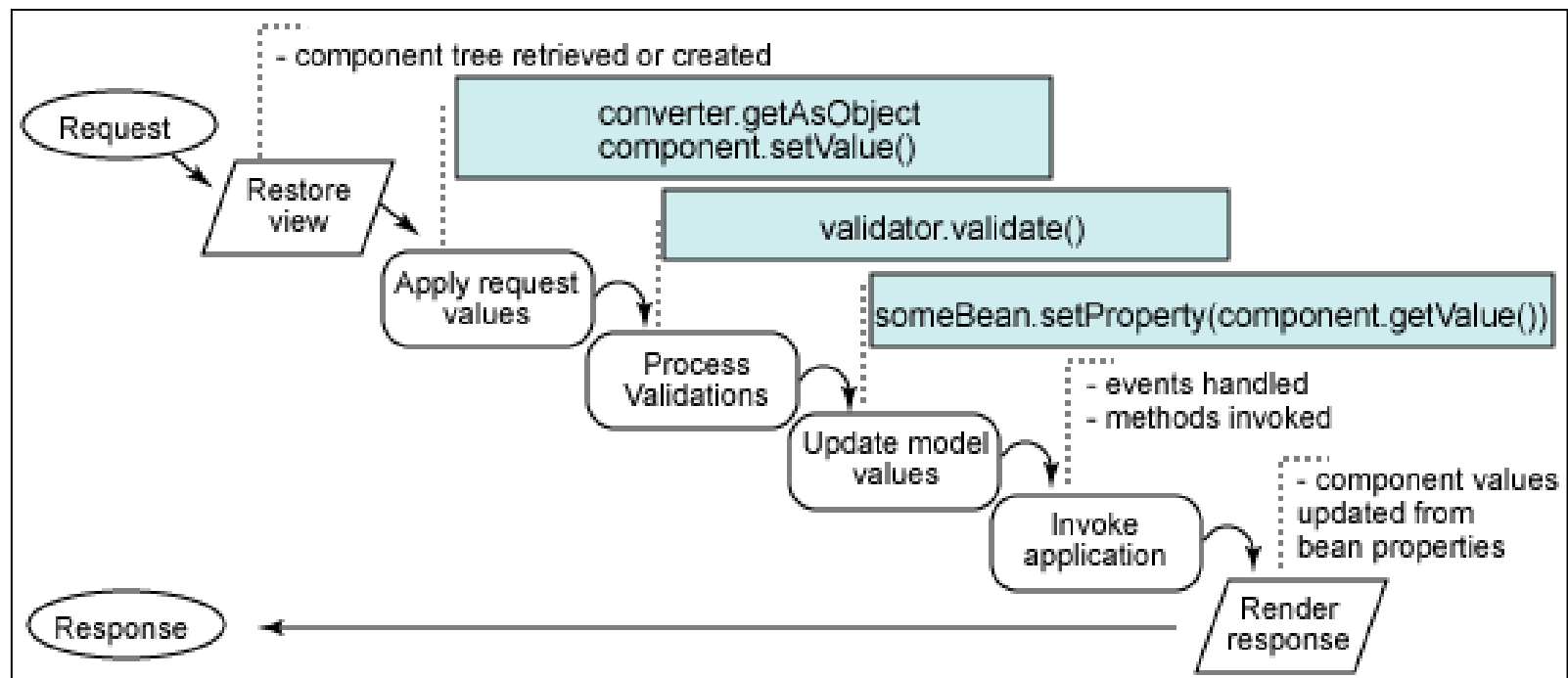
## ■ 5. Invoke application

- Nesta fase o controlador (controller) do JSF chama o método associado no submit, disparando assim a camada de regras de negócio da aplicação
- Todos os valores foram validados e carregados nas fases anteriores, por isso poderemos usá-los conforme necessitar.
  - Métodos ou “outcomes” carregados na propriedade “action” dos componentes



## ■ 6.Render Response

- Fase responsável por montar a resposta onde cada componente é processado e o resultado é unificado aos demais



## ■ PhaseListeners

- São componentes capazes de escutar as fases do ciclo de vida de uma requisição
- Devem implementar a interface `javax.faces.event.PhaseListener`
  - `public void beforePhase(PhaseEvent event);`
    - Será executado antes entrar na fase
  - `public void afterPhase(PhaseEvent event);`
    - Será executado após concluir a fase
  - `public PhaseId getPhaseId()`
    - Determina qual fase deverá ser escutada.
    - Valores possíveis estão na classe `"javax.faces.event.PhaseId"`

## ■ Exemplo: MyPhaseListener.java

```
public class MyPhaseListener implements PhaseListener {
 private static final long serialVersionUID = 7700487457990644826L;

 /** Metodo executado antes da fase. */
 public void beforePhase(PhaseEvent event) {
 System.out.println("Antes da fase: " + event.getPhaseId());
 }

 /** Metodo executado depois da fase. */
 public void afterPhase(PhaseEvent event) {
 System.out.println("Depois da fase: " + event.getPhaseId());
 }

 /** Metodo que indica qual fase sera "escutada". */
 public PhaseId getPhaseId() {
 return PhaseId.ANY_PHASE;
 }
}
```

## ■ Exemplo: faces-config.xml

```
<faces-config>
...
 <lifecycle>
 <phase-listener>
 br.com.cursojsf.listeners.MyPhaseListener
 </phase-listener>
 </lifecycle>
...
</faces-config>
```

## ■ Suporte ao método GET

### ■ View Parameters

- JSF 2 traz sanidade a esta situação com a introdução de “view parameters”, inspirado pelo JBoss Seam.
- Esses mapeamentos são especificados usando as novas tags `<f:viewParam>` e `<f:metadata>`:

```
<f:metadata>
```

```
 <f:viewParam name="nome" value="#{testeBean.nome}" />
```

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

- Podemos anexar qualquer conversor/validador a um componente `<f:viewParam>`, exatamente como faria com um `<h:inputText>`.

- Suporte ao método GET

- Evento PreRenderView

- JSF 2 inclui o PreRenderViewEvent que é disparado depois de ter terminado a carga de parâmetros e antes da página ser processada. Um listener pode ser registrado usando a tag <f:event>, por exemplo:

```
<f:metadata>
 <f:viewParam name="nome" value="#{testeBean.nome}" />
 <f:event type="preRenderView"
 listener="#{testeBean.gerarMensagem}" />
</f:metadata>
```

- O listener pode ser usado para carregar dados ou configurar o contexto necessário ao processamento da página.

## ■ Suporte ao método GET

### ■ <h:link>/<h:button>

- JSF 2 inclui dois novos componentes que simplificam a navegação GET: <h:link> e <h:button>.
- Estes componentes, geram a URL de destino.
- Utilizam o sistema de navegação do JSF para determinar a URL adequada.
- Em vez de indicar explicitamente a URL, podemos usar:

```
<h:link outcome="teste" value="Teste de GET">
 <f:param name="nome" value="Carlos" />
</h:link>
```

## ■ Suporte ao método GET

### ■ Exemplo: usuariosExibir.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:ui="http://java.sun.com/jsf/facelets"
 xmlns:h="http://java.sun.com/jsf/html"
 xmlns:f="http://java.sun.com/jsf/core">
<h:head>
 <title><h:outputText value="#{msg['page.usuarios.title']}" /></title>
</h:head>
<h:body>
 <f:metadata>
 <f:viewParam name="id" value="#{usuarioBean.usuario.id}" />
 <f:event type="preRenderView" listener="#{usuarioBean.exibir}" />
 </f:metadata>

 <h:panelGrid columns="2">
 <h:outputText value="#{msg['page.usuarios.label.id']}: " />
 <h:outputText value="#{usuarioBean.usuario.id}" />

 <h:outputText value="#{msg['page.usuarios.label.cpf']}: " />
 <h:outputText value="#{usuarioBean.usuario.cpf}">
 <f:converter converterId="converters.CpfConverter" />
 </h:outputText>
 </h:panelGrid>
</h:body>
</html>
```

...



- Suporte ao método GET
  - Exemplo: usuariosExibir.xhtml

...

```
<h:outputText value="#{msg['page.usuarios.label.nomeReduzido']}" />
<h:outputText value="#{usuarioBean.usuario.nomeReduzido}" />

<h:outputText value="#{msg['page.usuarios.label.nomeCompleto']}" />
<h:outputText value="#{usuarioBean.usuario.nomeCompleto}" />

<h:outputText value="#{msg['page.usuarios.label.email']}" />
<h:outputText value="#{usuarioBean.usuario.email}" />

<h:outputText value="#{msg['page.usuarios.label.dataNascimento']}" />
<h:outputText value="#{usuarioBean.usuario.dataNascimento}">
 <f:convertDateTime pattern="dd/MM/yyyy" />
</h:outputText>
</h:panelGrid>
<hr />
<h:button outcome="usuarios" value="#{msg['commons.voltar']}" />
</h:body>
</html>
```

# JSF 2.0 - Suporte ao método GET

- Suporte ao método GET
  - Exemplo: UsuarioBean.java

```
...
public void exhibir() {
 usuario = usuarioBusiness.selecionar(usuario);
}
...
```

- Exemplo: usuarios.xhtml

```
...
<h:column>
 <f:facet name="header">
 <h:outputText value="#{msg['commons.exibir']}" />
 </f:facet>
 <h:button outcome="usuariosExibir"
 value="#{msg['commons.exibir']}">
 <f:param name="id" value="#{usuario.id}" />
 </h:button>
</h:column>
...
```

## ■ Ajax

- A existência de diversos frameworks JSF/Ajax é uma boa indicação de que o JSF fornece uma base sólida para o Ajax.
- O que resta para a especificação JSF fazer?
  - Compatibilidade. Enquanto é impressionante que tantas soluções JSF/Ajax estejam disponíveis, as sutis diferenças entre essas soluções podem levar a problemas de compatibilidade.
  - O suporte nativo. Já era hora do JSF ter funcionalidade Ajax sem a necessidade de um framework de terceiros.

## ■ Ajax

### ■ jsf.ajax.request():

- API JavaScript primitiva que é dirigida principalmente para uso por frameworks, bem como pela implementação JSF em si.

```
<h:commandButton onclick="jsf.ajax.request(
 this, event, {render: 'foo'}); return false;"/>
```

### ■ <f:ajax>:

- JSF 2 inclui uma abordagem declarativa que se destina a ser mais conveniente para os desenvolvedores da página.

```
<h:commandButton>
 <f:ajax render="foo"/>
</h:commandButton>
```

## ■ Exemplo: number.xhtml

```
<!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:ui="http://java.sun.com/jsf/facelets"
 xmlns:h="http://java.sun.com/jsf/html" xmlns:f="http://java.sun.com/jsf/core">
<h:head>
 <title><h:outputText value="#{msg['page.number.title']}" /></title>
</h:head>
<h:body>
 <h:form id="numberForm">
 <h:outputLabel for="rangeField" value="#{msg['page.number.maximo']}: " />
 <h:inputText id="rangeField" value="#{numberBean.range}" />
 <h:commandButton action="#{numberBean.randomize}"
 value="#{msg['commons.novo']}">
 <f:ajax execute="@form" render="rangeField numField"
 onevent="concluido" onerror="erro" />
 </h:commandButton>

 <h:outputText value="#{msg['page.number.title']}: " />
 <h:outputText id="numField" value="#{numberBean.number}" /><hr />
 <h:commandButton action="index" value="#{msg['commons.voltar']}" />
 </h:form>
 <script type="text/javascript">
 function concluido(data) { alert("Evento: " + data.status); }
 function erro(data) { alert("Erro: " + data); }
 </script>
</h:body>
</html>
```

## ■ Exemplo: NumberBean.java

```
@ManagedBean
@ViewScoped
public class NumberBean {
 private double number = Math.random();
 private double range = 1.0;
 public double getRange() {
 return (range);
 }
 public void setRange(double range) {
 this.range = range;
 }
 public double getNumber() {
 return (number);
 }
 public void randomize() {
 number = range * Math.random();
 }
}
```

## ■ Exemplo: SystemMessages.properties

```
#Page: number
page.number.title=Número Randômico
page.number.maximo=Número máximo
```

## ■ Detalhes da TAG <f:ajax>

<f:ajax

```
execute="" // ID's dos inputs que serão processados
event="" // Evento JavaScript: use click ao invés de onclick
listener="" // Método do Bean que será executado
render="" // ID's dos componentes que serão recarregados
immediate="" // Pula a fase de validação
disabled="" // Habilita/Desabilita
onevent="" // Função JavaScript chamada nos eventos
onerror="" // Função JavaScript chamada quando houver erro
```

/>

- Nas propriedades execute e render, podemos usar: "@this", "@form", "@all", "@none" ou lista de ID's.

## ■ Exemplo: guess.xhtml

...

```
<h:inputText id="palpite"
 value="#{guessBean.palpite}"
 valueChangeListener="#{guessBean.onChange}">
 <f:ajax event="keyup" execute="@this"
 render="outtext btnEnviar" />
</h:inputText>
<h:commandButton id="btnEnviar"
 value="#{msg['page.guess.label.enviar']}"
 disabled="#{guessBean.palpite eq null or guessBean.palpite le 0}"
 action="#{guessBean.guess}" />
```

...



- The JEE(TM) 6.0 Tutorial
  - <http://download.oracle.com/javaee/6/tutorial/doc/>
- Core Servlets
  - <http://www.coreservlets.com/>
- JSF Tutorials
  - <http://www.jsftutorials.net/>
- Mkyong.com
  - <http://www.mkyong.com/tutorials/jsf-2-0-tutorials/>
- The Server Side
  - <http://www.theserverside.com/>
- IBM developerWorks
  - <http://www-128.ibm.com/developerworks>