



Desenvolvimento para Servidores-II

Java EE: uma visão geral

Neste tópico abordaremos os conceitos básicos de aplicações web e uma visão geral da tecnologia Java EE e uma introdução a JSF

Prof. Ciro Cirne Trindade

O que é uma aplicação web?

- "**Aplicação**" informalmente define um programa interativo
- Uma aplicação Web é um conjunto de páginas **interativas** que tem uma tarefa comum
- Está mais próximo do domínio geral de software do que de um simples "site" ou publicação online
- Usa tecnologia web como requisito, ao invés de programação *stand-alone*

Vantagens

- Poucos recursos locais
- Processamento centralizado
- Combina as vantagens do cliente-servidor com a facilidade de manutenção centralizada
- Compartilhamento de recursos

Por quê adotar?

- **Multiplataforma:** é possível usar a aplicação de qualquer navegador web
- Fácil de desenvolver: paradigma de páginas é simples; linguagens RAD; boa documentação
- Oferece um conjunto de **componentes** visuais **padronizados** com os quais o usuário está habituado
- Suporte a **interatividade** suficiente para a maior parte das aplicações
- **Portabilidade:** a aplicação muitas vezes é portátil entre servidores de múltiplas arquiteturas

Interação Usuário - Web

- O usuário pode interagir com o navegador de algumas formas diferentes:
 - Seguindo *hyperlinks* indicados no conteúdo
 - Interagindo com formulários
 - Gerando outros eventos DOM
 - Interagindo com um *plugin* embutido
- Das formas acima, as duas primeiras são as mais utilizadas quando se desenvolve uma aplicação web
- Seguir um *hyperlink* leva a uma outra URI

Parâmetros

- É possível passar variáveis na requisição
- Uma variável é simplesmente um **nome** acompanhado de um **dado**
- É fácil passar variáveis entre páginas
- Este mecanismo simples é a base de qualquer aplicação interativa:
 - manipulando links e elementos em uma página, passamos variáveis nomeadas para o servidor, que as trata de forma especial

Formulários

- Visivelmente, a interação que se pode ter com links é limitada: só podemos seguir links pré-elaborados e não há riqueza de interação
- Para resolver este problema, foram padronizados **formulários web**

Formulários

- Um formulário é uma seção de página que contém componentes de interface interativos

Caixa de entrada (text):	<input type="text"/>
Botão Envio (submit):	<input type="submit" value="Enviar Formulário"/>
Botão Rádio (radio):	<input type="radio"/> <input type="radio"/> <input type="radio"/>
Botão seleção (checkbox):	<input type="checkbox"/>
Lista de seleção (select e option):	<input type="text" value="Pedra"/> ▼
Envio de arquivo (file)	<input type="text"/> <input type="button" value="Procurar..."/>
Área de texto (textarea)	<div><div></div><div>▲▼</div></div>

Plataforma Java EE (1/2)

- Conjunto de recursos disponibilizados em um servidor Java EE via protocolo HTTP
 - Documentos estáticos (páginas HTML, imagens, etc.)
 - Servlets e outras classes executadas no servidor
 - Páginas JSP/JSF
 - Informações de configuração
 - Classes que são armazenadas no servidor mas são executadas no cliente (Applets)
 - Scripts

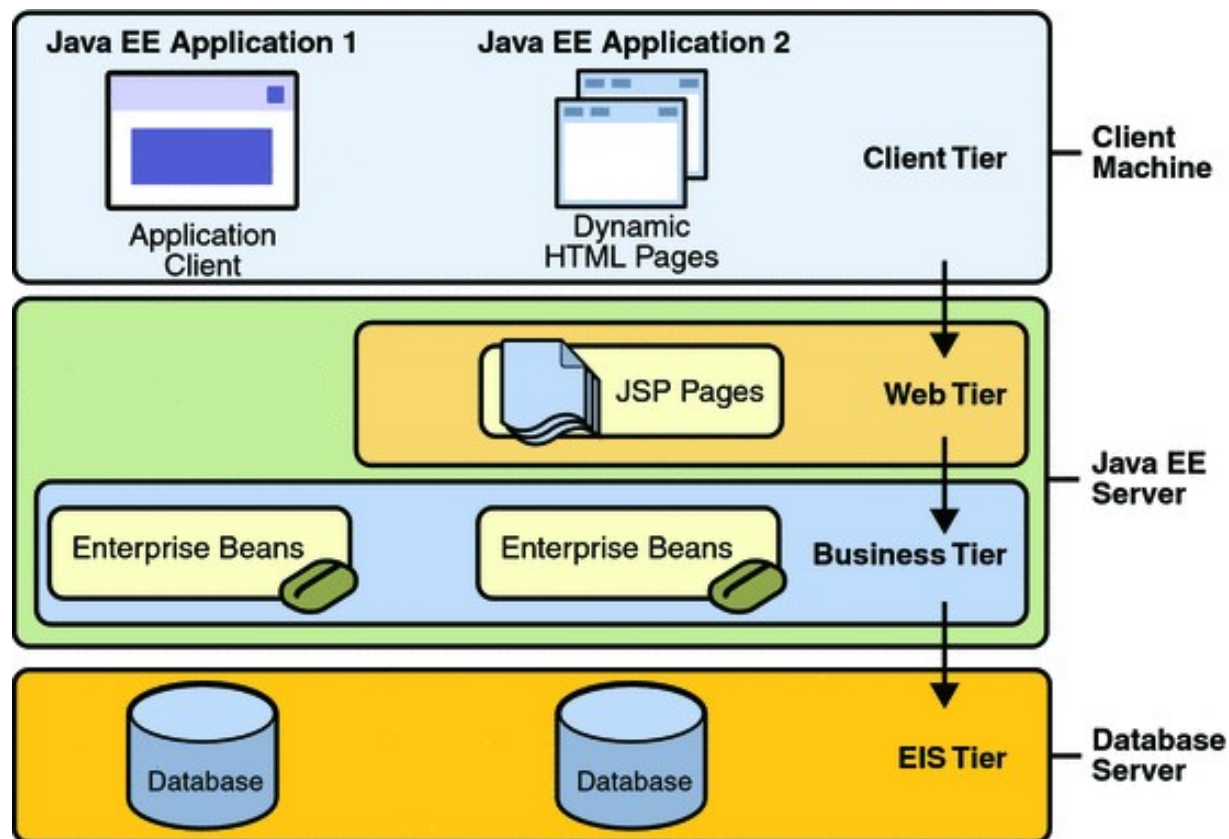
Plataforma Java EE (2/2)

- Conjunto de APIs que estendem a plataforma Java para atender às demandas da computação corporativa
 - JavaServer Pages (JSP)
 - Java Servlets
 - JavaServer Pages Standard Tag Library (JSTL)
 - JavaServer Faces (JSF)
 - Enterprise JavaBeans (EJB)
 - Java Persistence API (JPA)
 - Java Message Service (JMS)
 - Java Naming and Directory Interface (JNDI)
 - Entre outras

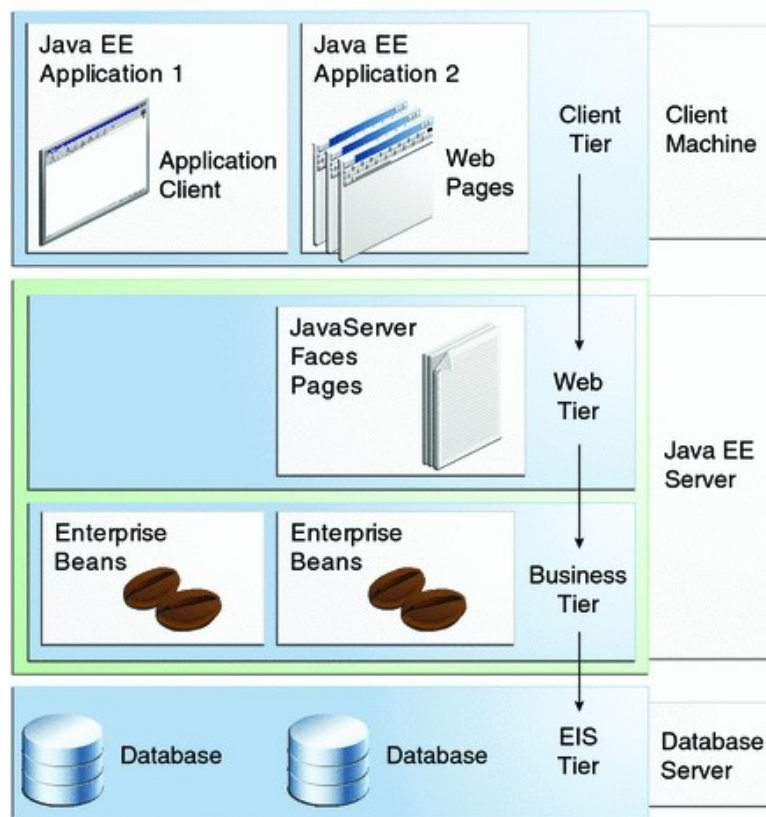
Histórico

- J2EE 1.4 (2003)
 - Aplicações web baseadas principalmente em Servlets e JSP
 - Introduz JSF
- Java EE 5 (2006)
 - Introduz JPA para persistência e anotações para substituir os arquivos de configuração baseados em XML
- Java EE 6 (2009)
 - Consolida JSF e JPA
- Java EE 7 (2013)
 - Suporte a HTML5

Aplicações distribuídas multicamadas (Java EE 5)



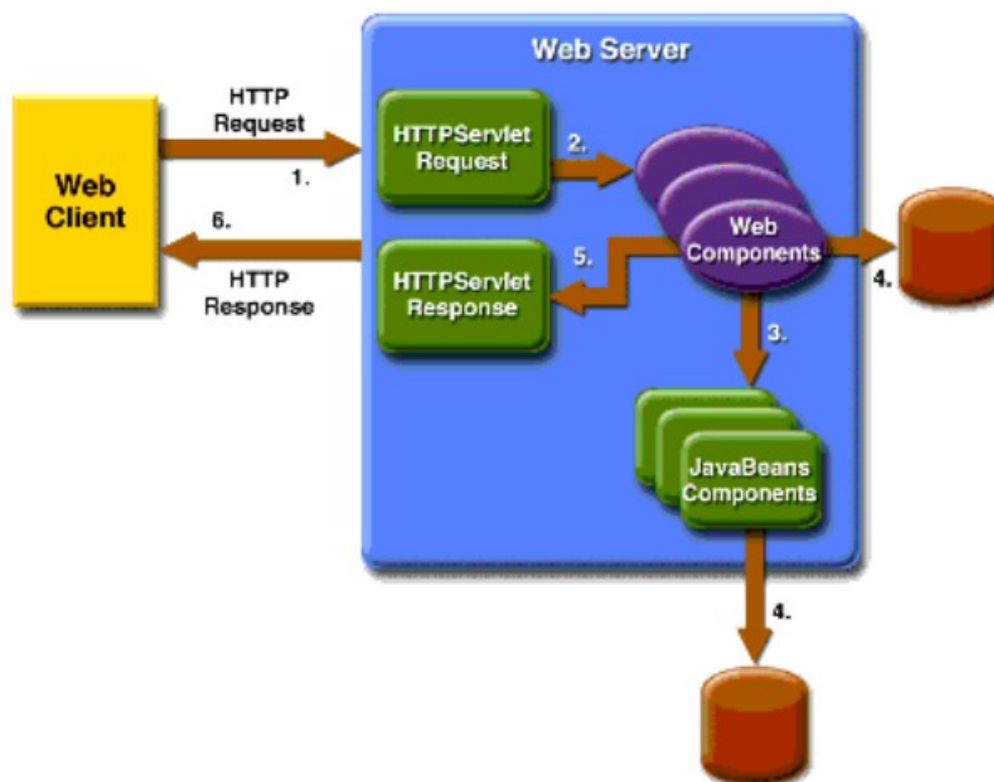
Aplicações distribuídas multicamadas (Java EE 6)



Acesso a aplicações Java EE

- O cliente, em geral um navegador web, gera um HTTP request que é enviado para uma aplicação Web que reside em um servidor Java EE (por exemplo, Tomcat)
- O servidor Java EE converte o HTTP request em um objeto HTTP request e o envia ao componente da aplicação Web identificado pela URL contida no HTTP request
- O componente cria um HTTP response, preenchendo-o com a resposta a solicitação feita e o envia ao cliente

Acesso a aplicações Java EE



Servidor de aplicação (1/2)

- Java EE é um grande conjunto de especificações
- Existem diversas implementações dessas especificações
- Um servidor de aplicação Java EE é uma implementação completa dessas especificações

Servidor de aplicação (2/2)

- Principais servidores de aplicação Java EE
 - JBoss (RedHat), gratuito
 - GlassFish (Oracle), gratuito
 - Geronimo (Apache), gratuito
 - WebSphere (IBM)

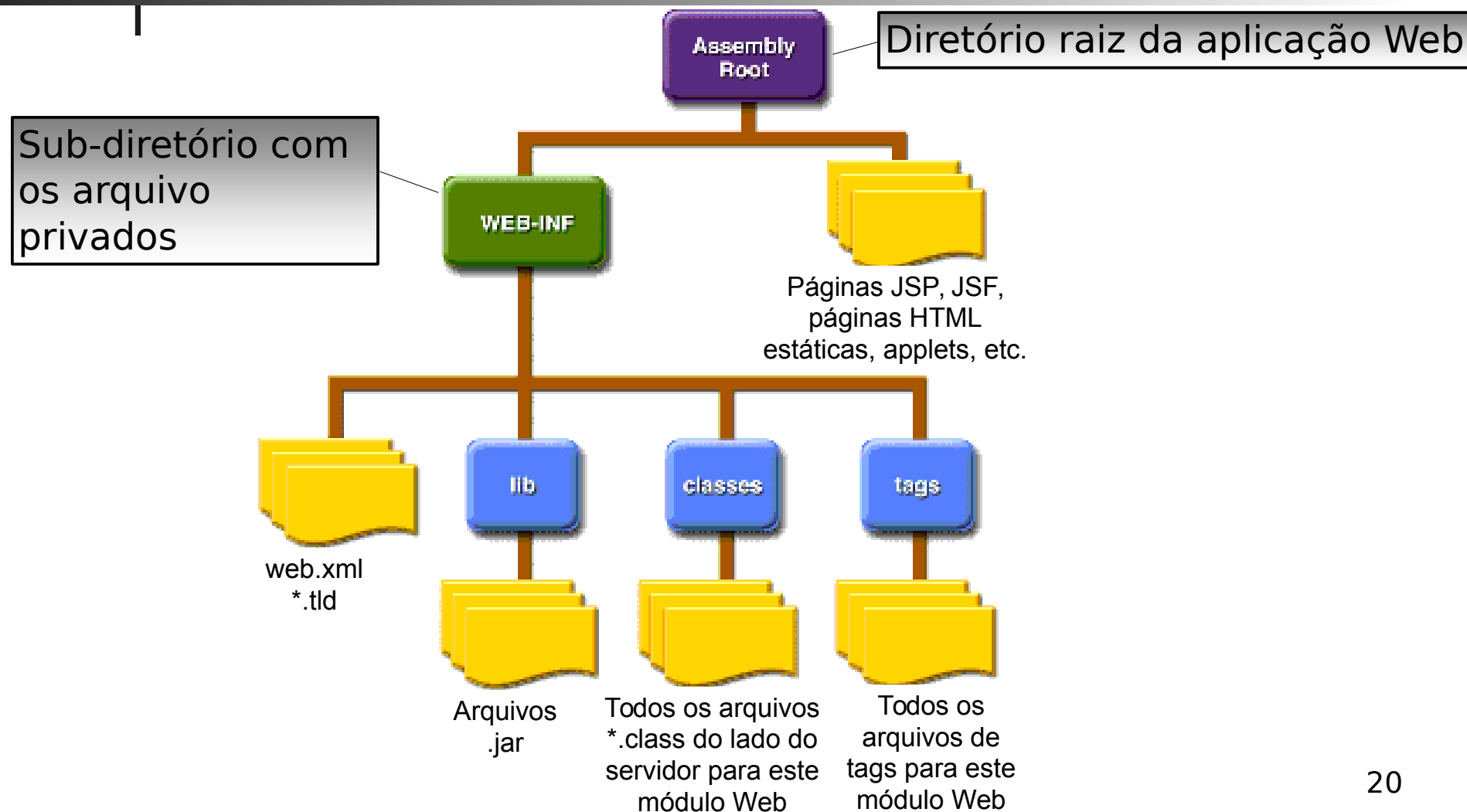
Web container (1/2)

- Um web container é um servidor que suporta apenas as especificações para lidar com o desenvolvimento de uma aplicação web:
 - JSP
 - Servlets
 - JSTL
 - JSF

Web container (2/2)

- Principais web containers Java EE:
 - Tomcat (Apache), gratuito
 - Jetty (Codehaus), gratuito

Estrutura de diretório da Aplicação Web



Descritor

- Em Java EE todo descritor segue o padrão XML e chama-se **web.xml**
- Através web.xml é possível:
 - Configurar o ambiente de execução da aplicação web
 - Configurar a aplicação web (ativação, recursos, parâmetros, ...)
 - Definir página inicial e páginas de erro
 - Fazer mapeamento entre URLs com Servlets e JSPs
 - Definir aspectos de segurança

JSF (JavaServer Faces)

- JSF é um *framework* baseado em componentes para construção da interface com o usuário
- A tecnologia JSF consiste em:
 - Uma API para representar componentes e gerenciar seu estado
 - Tag libraries para adicionar componentes às páginas web e conectá-los a objetos no servidor

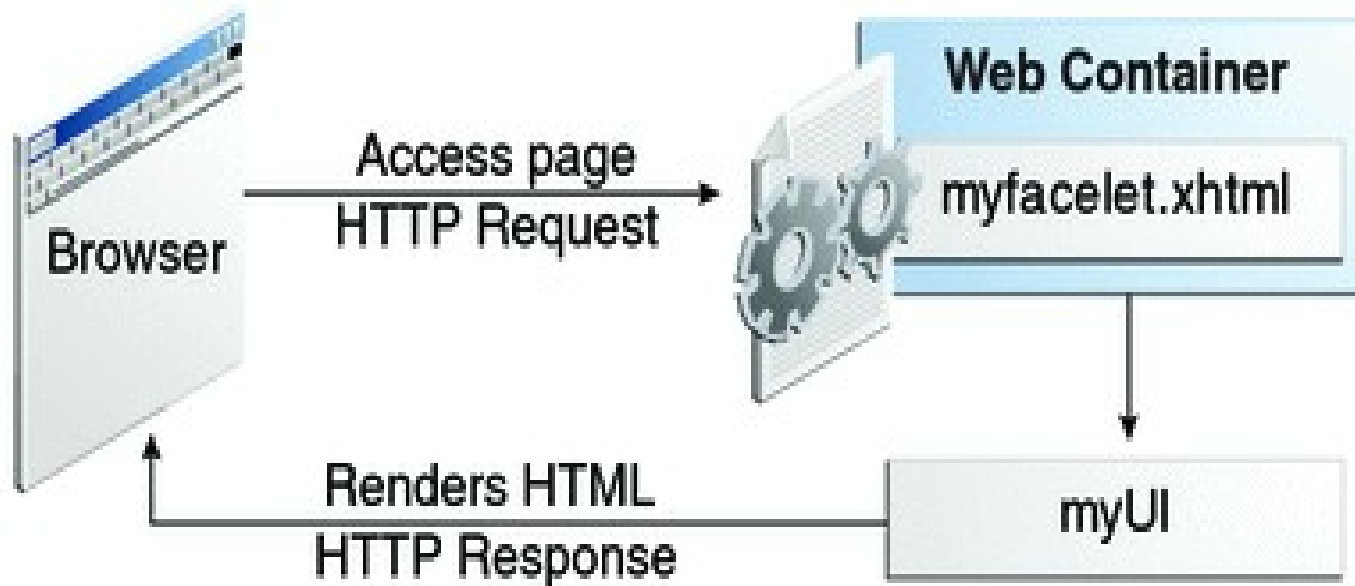
Benefícios da tecnologia JSF

- Separação clara entre lógica da aplicação e apresentação
- Mapeia requisições HTTP em componentes específicos e gerencia esses componentes como objetos no servidor

O que é uma aplicação JSF?

- Uma aplicação JSF típica inclui as seguintes partes
 - Um conjunto de páginas web aonde os componentes residem
 - Um conjunto de tags para adicionar componentes à página
 - Um conjunto de *managed beans* (POJO's)

Interação entre o cliente e o servidor em uma aplicação JSF



Configurando uma página JSF (1/3)

- Uma página JSF geralmente inclui os seguintes elementos:
 - Um conjunto de declarações de namespaces que declaram as tag libraries JSF
 - Opcionalmente, as novas tags HTML `head` (`h:head`) e `body` (`h:body`)
 - Uma tag `form` (`h:form`) que representa os componentes de entrada do usuário

Configurando uma página JSF (2/3)

- Para acrescentar componentes JSF a sua página, você precisa fornecer acesso para as duas tag libraries padrão:
 - JSF HTML tag library
 - Define tags que representam componentes de interface HTML
(<https://docs.oracle.com/javaee/7/javaxserver-faces-2-2/renderkitdocs/toc.htm>)
 - JSF core tag library
 - Define tags que representam ações básicas
(<https://docs.oracle.com/javaee/7/javaxserver-faces-2-2/vdldocs-jsp/toc.htm>)

Configurando uma página JSF (3/3)

- Para usar as tags JSF é necessário incluir diretivas apropriadas no início da página XHTML
- ```
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:h="http://xmlns.jcp.org/jsf/html"
 xmlns:f="http://xmlns.jcp.org/jsf/core"
 >
```

# JSF HTML tag library

---

- Possui componentes para exibir dados ou aceitar dados do usuário
- Os dados são coletados como parte de um formulário e submetidos ao servidor, geralmente quando o usuário clica em um botão

## A tag `h:form` (1/2)

- Aplicações web acontecem através de submissão de formulários
- A tag `h:form` do JSF é equivalente a tag `form` do HTML, porém não possui os atributos `method` e `action`
  - Todas as submissões de formulário em JSF são feitas através do método POST
  - As submissões de formulários JSF são postadas para a página atual

## A tag `h:form` (2/2)

- Uma tag `h:form` representa um formulário de entrada de dados, que inclui componentes filhos que podem conter dados que tanto podem ser apresentados ao usuário ou submetidos com o formulário

# A tag `h:panelGrid` (1/2)

- Pode ser usada no lugar de uma tabela HTML
- Para especificar o número de colunas utilize o atributo `columns` (o valor default é 1)

- Exemplo:

```
<h:panelGrid columns="3">
```

```
...
```

```
</h:panelGrid>
```

Organiza os elementos  
em colunas da esquerda  
para a direita



## A tag `h:panelGrid` (2/2)

- Para agrupar dois ou mais elementos em uma célula de um `h:panelGrid` normalmente utiliza-se a tag `h:panelGroup`

- Exemplo:

```
<h:panelGrid columns="2">
 ...
 <h:panelGroup>
 <h:inputText id="name" value="#{user.name}">
 <h:message for="name"/>
 </h:panelGroup>
 ...
</h:panelGrid>
```

Agrupar o `inputText` e o `message` em uma única célula

# As tags `h:inputText` e `h:outputText`

- A tag `h:inputText` é usada para exibir uma caixa de entrada de texto
  - A maioria das tags de entrada do JSF possui um atributo booleano chamado `required` que permite indicar que seu valor é obrigatório
  - Neste caso, o atributo `requiredMessage` pode ser usado para definir uma mensagem de advertência
- A tag `h:outputText` é semelhante, mas exibe um texto somente leitura em uma única linha

# A tag `h:commandButton` (1/2)

- Se você utiliza uma tag `h:commandButton`, os dados da página atual são processados quando um usuário clica no botão

- Por exemplo:

```
<h:commandButton value="Submit"
action="#{cashier.submit}"/>
```

- O atributo `value` representa o rótulo do botão

## A tag `h:commandButton` (2/2)

- O atributo `action` faz uma das seguintes coisas:
  - Especifica uma `String` que diz a aplicação que página acessar na sequência
  - Referencia um método de um *managed bean* que executa algum processamento e devolve uma `String`

# Criando uma aplicação JSF simples (1/5)

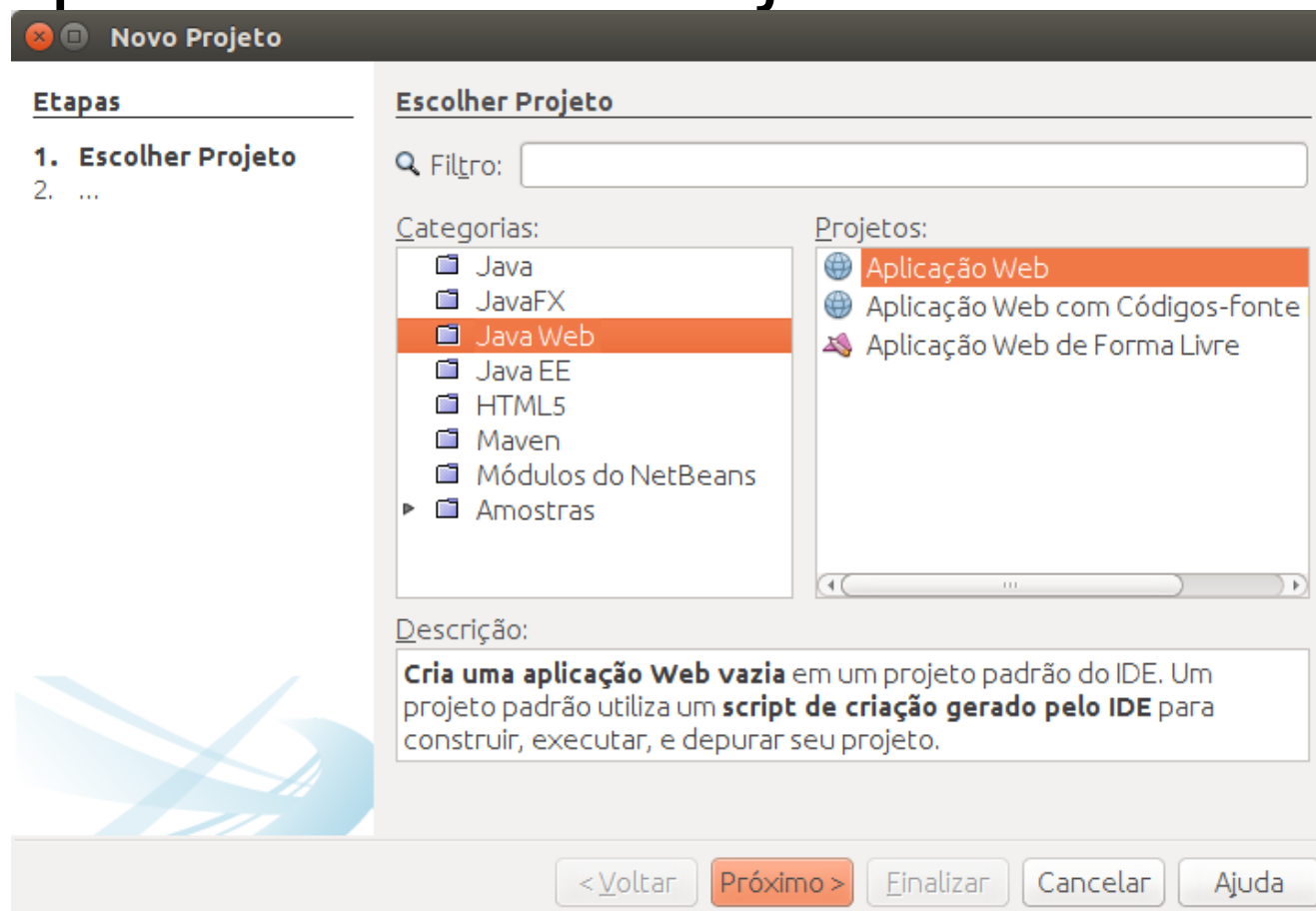
- Desenvolver uma aplicação JSF geralmente requer as seguintes tarefas:
  - Implementar *managed beans*
  - Criar páginas web usando tags de componentes
  - Mapear a instância `FacesServlet`

# Criando uma aplicação JSF simples (2/5)

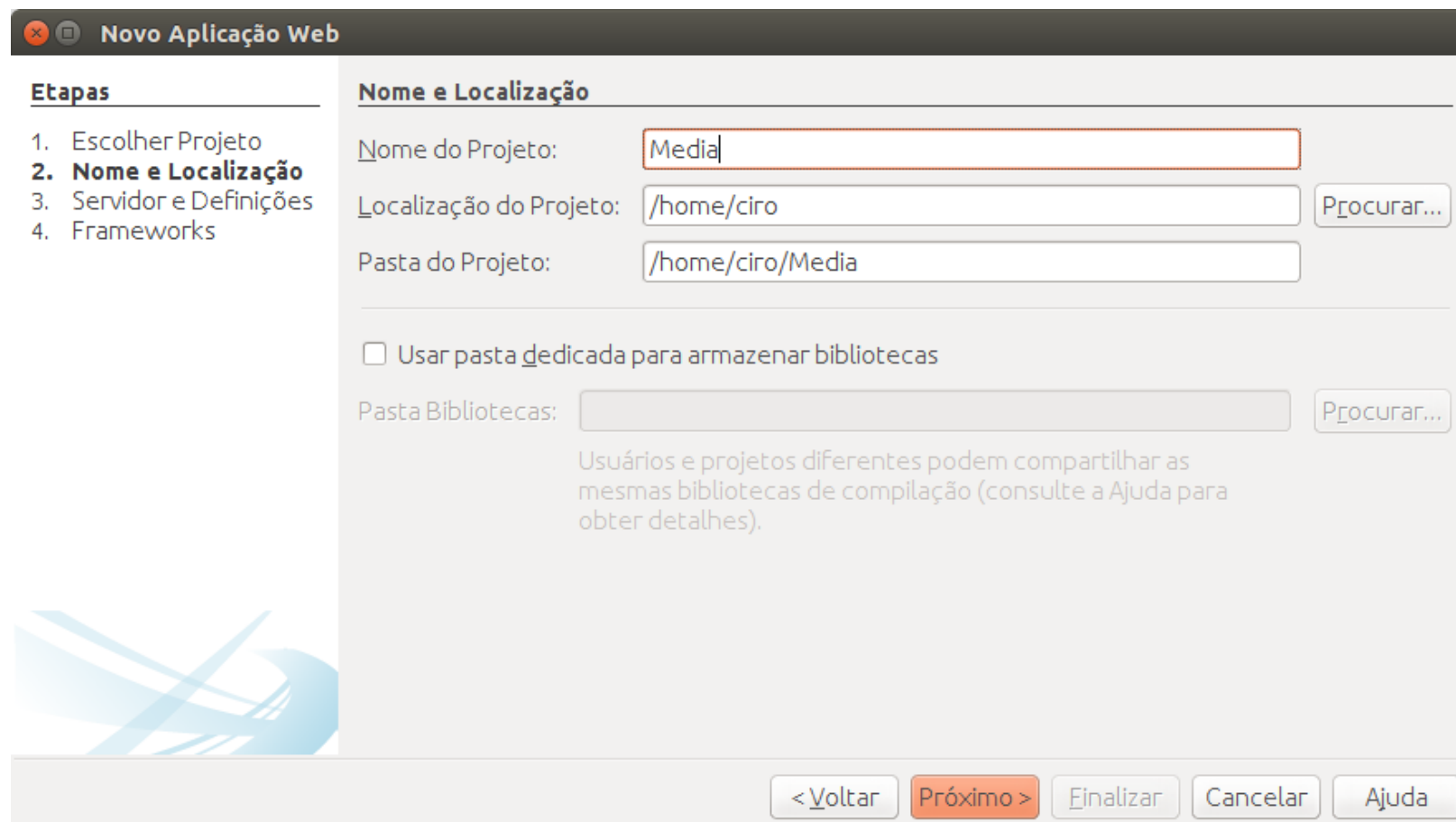
- Este exemplo é uma aplicação que calcula a média das notas de um aluno e determina sua situação
- Consiste em um *managed bean* e duas página web
- O usuário deve preencher um formulário com seu nome e as notas de duas provas e submeter essas informações ao servidor que deverá calcular a média do aluno, definir sua situação e exibir essas informações ao cliente

# Criando uma aplicação web na IDE NetBeans (1/4)

- Arquivo → Novo Projeto



# Criando uma aplicação web na IDE NetBeans (2/4)



**Novo Aplicação Web**

**Etapas**

1. Escolher Projeto
- 2. Nome e Localização**
3. Servidor e Definições
4. Frameworks

**Nome e Localização**

Nome do Projeto:

Localização do Projeto:

Pasta do Projeto:

☐ Usar pasta dedicada para armazenar bibliotecas

Pasta Bibliotecas:

Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes).

< Voltar **Próximo >** Finalizar Cancelar Ajuda



# Criando uma aplicação web na IDE NetBeans (3/4)

**Novo Aplicação Web**

**Etapas**

1. Escolher Projeto
2. Nome e Localização
- 3. Servidor e Definições**
4. Frameworks

**Servidor e Definições**

Adicionar à aplicação corporativa: <Nenhum(a)>

Servidor: GlassFish Server 4.1

Versão do Java EE: Java EE 7 Web

Caminho do Contexto: /Media

< Voltar Próximo > Finalizar Cancelar Ajuda

# Criando uma aplicação web na IDE NetBeans (4/4)

**Novo Aplicação Web**

**Etapas**

1. Escolher Projeto
2. Nome e Localização
3. Servidor e Definições
- 4. Frameworks**

**Frameworks**

Selecione os frameworks que você deseja utilizar em sua aplicação Web.

- ☐ Spring Web MVC
- ☒ **JavaServer Faces**
- ☐ Struts 1.3.10
- ☐ Hibernate 4.3.1

**Configuração JavaServer Faces**

Bibliotecas | **Configuração** | Componentes

☒ **Biblioteca do Servidor:** JSF 2.2

☐ Bibliotecas Registradas: JSF 2.2

☐ **Criar Nova Biblioteca**

JAR ou Pasta JSF:  Procurar...

Nome da Biblioteca:

< Voltar Próximo > **Finalizar** Cancelar Ajuda

# Criando uma aplicação JSF simples (3/5)

- Implementando o *managed bean*
  - Componentes em uma página são associados a *managed beans* que provêm a lógica da aplicação

# Aluno.java (1/3)

```
package beans;
```

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
```

```
@ManagedBean
@RequestScoped
public class Aluno {
 private String nome;
 private Float p1;
 private Float p2;
 private Float media;
 private String situacao;

 public Aluno() { }
```

Registra o *managed bean* como um recurso da aplicação JSF

# Aluno.java (2/3)

```
public String getNome() { return nome; }
public void setNome(String nome) {
 this.nome = nome;
}
public Float getP1() { return p1; }
public void setP1(Float p1) {
 this.p1 = p1;
}
public Float getP2() { return p2; }
public void setP2(Float p2) {
 this.p2 = p2;
}
public Float getMedia() { return media; }
public String getSituacao() { return situacao; }
```

# Aluno.java (3/3)

```
public String calcularMedia() {
 media = (p1 + p2) / 2;
 if (media >= 6) {
 situacao = "Aprovado";
 }
 else if (media >= 3) {
 situacao = "Substitutiva";
 }
 else {
 situacao = "Reprovado";
 }
 return "/media";
}
```

O valor de retorno deste método referencia a página que deve ser renderizada após sua execução

# Criando uma aplicação JSF simples (4/5)

- Criando a página web
  - Normalmente as página web em uma aplicação JSF são criadas em XHTML
  - A página web se conecta ao *managed bean* através de Expression Language (EL) no formato:
    - `#{managedbean.atributo}`

# index.xhtml

```
<?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">
 <h:head>
 <title>Cálculo da Média</title>
 </h:head>
 <h:body>
 <h:form>
 <h:panelGrid columns="2">
 <h:outputText value="Nome:"/>
 <h:inputText value="#{aluno.nome}"/>
 <h:outputText value="Nota da 1ª prova:"/>
 <h:inputText value="#{aluno.p1}" required="true"/>
 <h:outputText value="Nota da 2ª prova:"/>
 <h:inputText value="#{aluno.p2}" required="true"/>
 </h:panelGrid>
 <h:commandButton value="Calcular média"
 action="#{aluno.calcularMedia}"/>
 </h:form>
 </h:body>
</html>
```

Se nenhum nome for definido na anotação @ManagedBean, o *managed bean* pode ser acessado com a 1ª letra do nome da classe em minúsculo

Referencia o método a ser executado na submissão do formulário



# media.xhtml

```
<?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">
 <h:head>
 <title>Média</title>
 </h:head>
 <h:body>
 Nome: <h:outputText value="#{aluno.nome}"/>

 Média: <h:outputText value="#{aluno.media}"/>

 Situação: <h:outputText
 value="#{aluno.situacao}"/>

 <h:button outcome="index" value="Voltar"/>
 </h:body>
</html>
```

# Criando uma aplicação JSF simples (5/5)

- Mapeando a instância `FacesServlet`
  - A tarefa final é mapear o `FacesServlet`, o que é feito através do `web.xml`

# Fragmento do web.xml

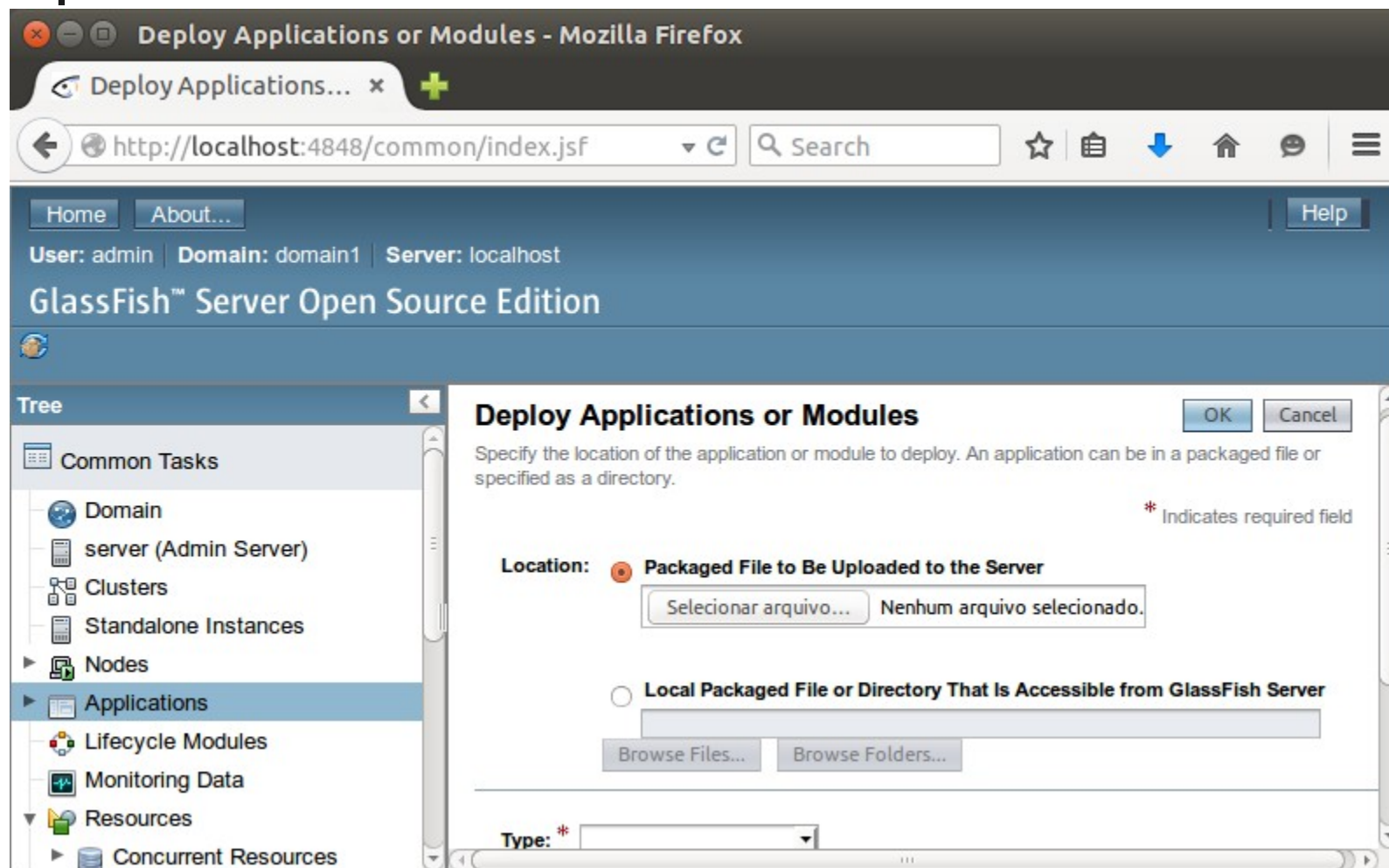
Indica a ordem que o servlet é inicializado, 1 indica que este é o 1º

```
...
<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>/faces/*</url-pattern>
</servlet-mapping>
...
```

# Web Application Archieves (WAR)

- Para fazer o *deployment* (implantação) de uma aplicação web o mais comum é empacotar todo o conteúdo do diretório da aplicação web em um arquivo do tipo **war**
- Para gerar o war no NetBeans selecione a opção Executar→Criar Projeto (F11) ou Executar→Limpar e Construir Projeto (Shift+F11)

# Disponibilização da aplicação no Glassfish



# A tag `h:inputSecret`

- A tag `h:inputSecret` **exibe** uma tag HTML `<input type="password">`

- **Por exemplo:**

```
<h:inputSecret redisplay="false"
value="#{loginBean.password}" />
```

- Neste exemplo, o atributo `redisplay` foi definido como `false`, isto impede que a senha seja exibida em uma *query string* ou no arquivo fonte da página HTML resultante

# Exemplo de uma aplicação JSF para autenticação

- A aplicação consistirá dos seguintes recursos:
  - *Managed bean*: SimpleLogin.java
  - Páginas web: login.xhtml, success.xhtml e fail.xhtml
  - Arquivo de configuração: web.xml



# SimpleLogin.java (1/2)

```
package beans;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean (name="user")
@SessionScoped
public class SimpleLogin {
 private static final String LOGIN = "ciro";
 private static final String PASSWORD = "123";
 private String loginname;
 private String password;

 public SimpleLogin() { }

 public String getLoginname() { return loginname; }

 public void setLoginname(String loginname) {
 this.loginname = loginname;
 }
}
```



# SimpleLogin.java (1/2)

```
public String getPassword() {
 return password;
}

public void setPassword(String password) {
 this.password = password;
}

public String validar() {
 if (loginname.equals(LOGIN)
 && password.equals(PASSWORD)) {
 return "/success";
 }
 return "/fail";
}
}
```

# login.xhtml

```
<?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">
 <h:head><title>Login Facelet</title></h:head>
 <h:body>
 <h:form>
 <h:panelGrid columns="2">
 <h:outputText value="Login: " />
 <h:inputText id="loginname"
 value="#{user.loginname}" required="true"
 requiredMessage="Erro: informe o login"/>
 <h:outputText value="Senha: " />
 <h:inputSecret id="password" redisplay="false"
 value="#{user.password}" required="true"
 requiredMessage="Erro: informe a senha"/>
 <h:commandButton value="Login" action="#{user.validar}"/>
 </h:panelGrid>
 </h:form>
 </h:body>
</html>
```

# success.xhtml

```
<?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">
 <h:head>
 <title>Sucesso Facelet</title>
 </h:head>
 <h:body>
 <h1>Sucesso! Bem-vindo a aplicação JSF,
 #{user.loginname}</h1>
 </h:body>
</html>
```

# fail.xhtml

```
<?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">
 <h:head>
 <title>Falha Facelet</title>
 </h:head>
 <h:body>
 <h1>Login/senha inválidos!</h1>
 </h:body>
</html>
```

# web.xml (1/2)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
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-app_3_0.xsd">
 <context-param>
 <param-name>javax.faces.PROJECT_STAGE
 </param-name>
 <param-value>Development</param-value>
 </context-param>
 <servlet>
 <servlet-name>Faces Servlet</servlet-name>
 <servlet-class>javax.faces.webapp.FacesServlet
 </servlet-class>
 <load-on-startup>1</load-on-startup>
 </servlet>
```

# web.xml (2/2)

```
<servlet-mapping>
 <servlet-name>Faces Servlet</servlet-name>
 <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
<session-config>
 <session-timeout>
 30
 </session-timeout>
</session-config>
<welcome-file-list>
 <welcome-file>faces/login.xhtml</welcome-file>
</welcome-file-list>
</web-app>
```

# Tags de Seleção

- JSF possui 7 tags para fazer seleções:
  - `h:selectBooleanCheckbox`
  - `h:selectManyCheckbox`
  - `h:selectOneRadio`
  - `h:selectOneListbox`
  - `h:selectManyListbox`
  - `h:selectOneMenu`
  - `h:selectManyMenu`

# h:selectBooleanCheckbox

- Caixa de seleção simples, equivalente a tag `<input type="checkbox">` do HTML
- Pode ser associado a um atributo booleano de um managed bean
- Exemplo:

Entre em contato:

```
<h:selectBooleanCheckbox
 value="#{user.contato}"/>
```



# h:selectManyCheckbox (1/3)

- Cria um grupo de caixas de seleção a partir de onde pode-se selecionar uma ou mais caixas de seleção do grupo
- O grupo é definido no corpo do `h:selectManyCheckbox` através de uma tag `f:selectItems` ou várias tags `f:selectItem`

## h:selectManyCheckbox (2/3)

### ■ Exemplo: seleção de cores

```
<h:selectManyCheckbox value="#{user.cores}">
 <f:selectItem itemValue="Vermelho"
 itemLabel="Vermelho"/>
 <f:selectItem itemValue="Azul"
 itemLabel="Azul"/>
 <f:selectItem itemValue="Amarelo"
 itemLabel="Amarelo"/>
 <f:selectItem itemValue="Verde"
 itemLabel="Verde"/>
</h:selectManyCheckbox>
```

## h:selectManyCheckbox (3/3)

- É possível acrescentar uma borda ao grupo de caixas de seleção através do atributo `border`, que especifica a largura da borda
- O atributo `layout` pode ser usado para definir a orientação do grupo de caixas de seleção
  - O valor desse atributo pode ser `lineDirection` (horizontal – default) ou `pageDirection` (vertical)

## h:selectOneRadio (1/2)

- Botões de seleção única, equivalente a tag `<input type="radio">` do HTML
- Exemplo: seleção da escolaridade

```
<h:selectOneRadio value="#{user.educacao}">
 <f:selectItem itemValue="Ensino Fundamental"
 itemLabel="Ensino Fundamental"/>
 <f:selectItem itemValue="Ensino Médio"
 itemLabel="Ensino Médio"/>
 <f:selectItem itemValue="Graduação"
 itemLabel="Graduação"/>
 <f:selectItem itemValue="Pós-graduação"
 itemLabel="Pós-graduação"/>
</h:selectOneRadio>
```

## `h:selectOneRadio` (2/2)

- O atributo `value` define que botão está selecionado
- Assim como a tag `h:selectManyCheckbox`, a tag `h:selectOneRadio` também possui os atributos `border` e `layout`

# `h:selectOneListbox` e `h:selectManyListBox`

---

- Listas de seleção equivalentes a tag `<select>` do HTML
- Os itens das listas são definidos no corpo das tags `h:selectOneListbox` e `h:selectManyListbox` por tags `f:selectItem`
- `h:selectOneListbox` permite selecionar um item da lista, enquanto `h:selectManyListbox` permite a seleção de mais de um item da lista

# Exemplo do

## h:selectOneListbox: seleção do ano

```
<h:selectOneListbox value="#{user.ano}"
 size="5">
```

```
<f:selectItems
```

```
 values="#{user.anosEducacao}"
```

```
</h:selectOneListbox>
```

Define o  
número de  
itens visíveis

```
public String[] getAnosEducacao() {
 String[] anos = new String[20];
 int anoAtual = new GregorianCalendar().
 get(GregorianCalendar.YEAR);
 for (int i = 0; i < 20; i++)
 anos[i] = String.valueOf(anoAtual - i);
 return anos;
}
```

# Exemplo do

## h:selectManyListbox: seleção dos idiomas

```
<h:selectManyListbox value="#{user.idiomas}">
 <f:selectItem itemValue="Inglês"
 itemLabel="Inglês"/>
 <f:selectItem itemValue="Francês"
 itemLabel="Francês"/>
 <f:selectItem itemValue="Espanhol"
 itemLabel="Espanhol"/>
 <f:selectItem itemValue="Italiano"
 itemLabel="Italiano"/>
</h:selectManyListbox>
```



`h:selectOneMenu` e  
`h:selectManyMenu`

---

- São semelhantes às tags  
`h:selectOneListbox` e  
`h:selectManyListbox`, a diferença é  
que os itens são exibidos como um  
menu suspenso

# Referências

- ORACLE Corporation. *The Java EE 7 Tutorial*. Disponível em: <https://docs.oracle.com/javaee/7/JEETT.pdf>, 2014.
- GEARY, David; HORSTMANN, Cay. *Core JavaServer Faces*. 3. ed., Prentice Hall, 2010.