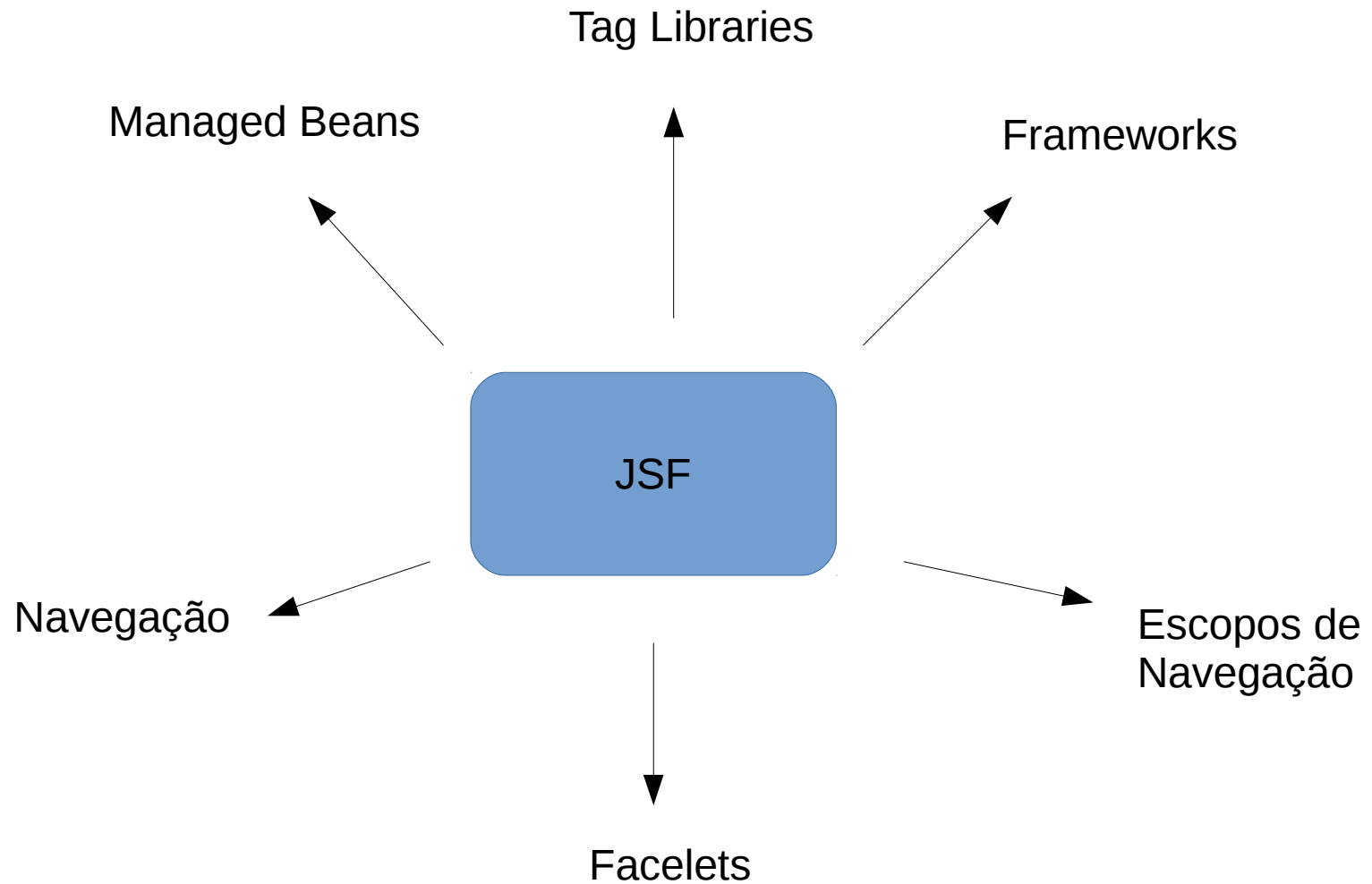
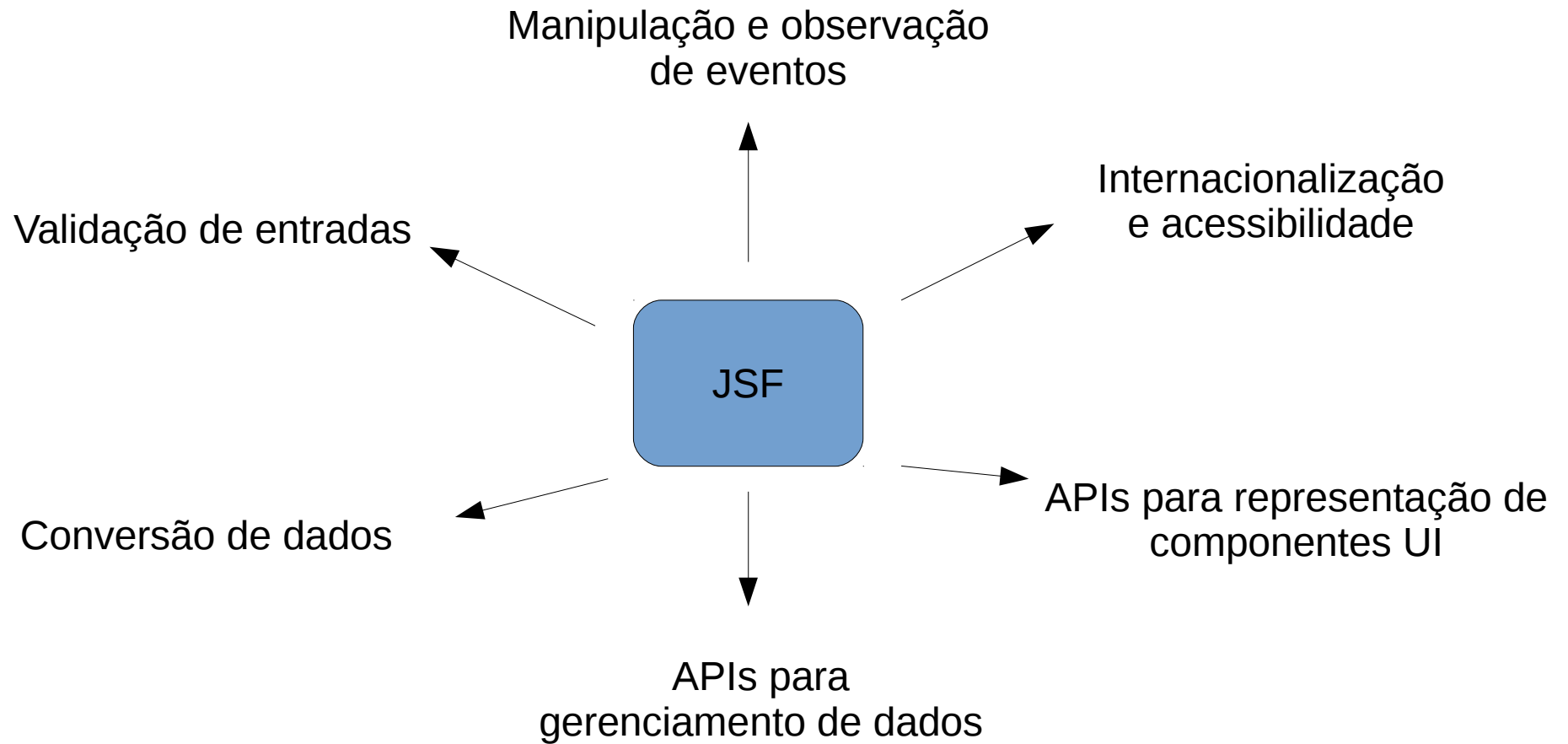


# OPÇÃO 1



## OPÇÃO 2

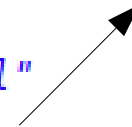


## TagLibs do JSF

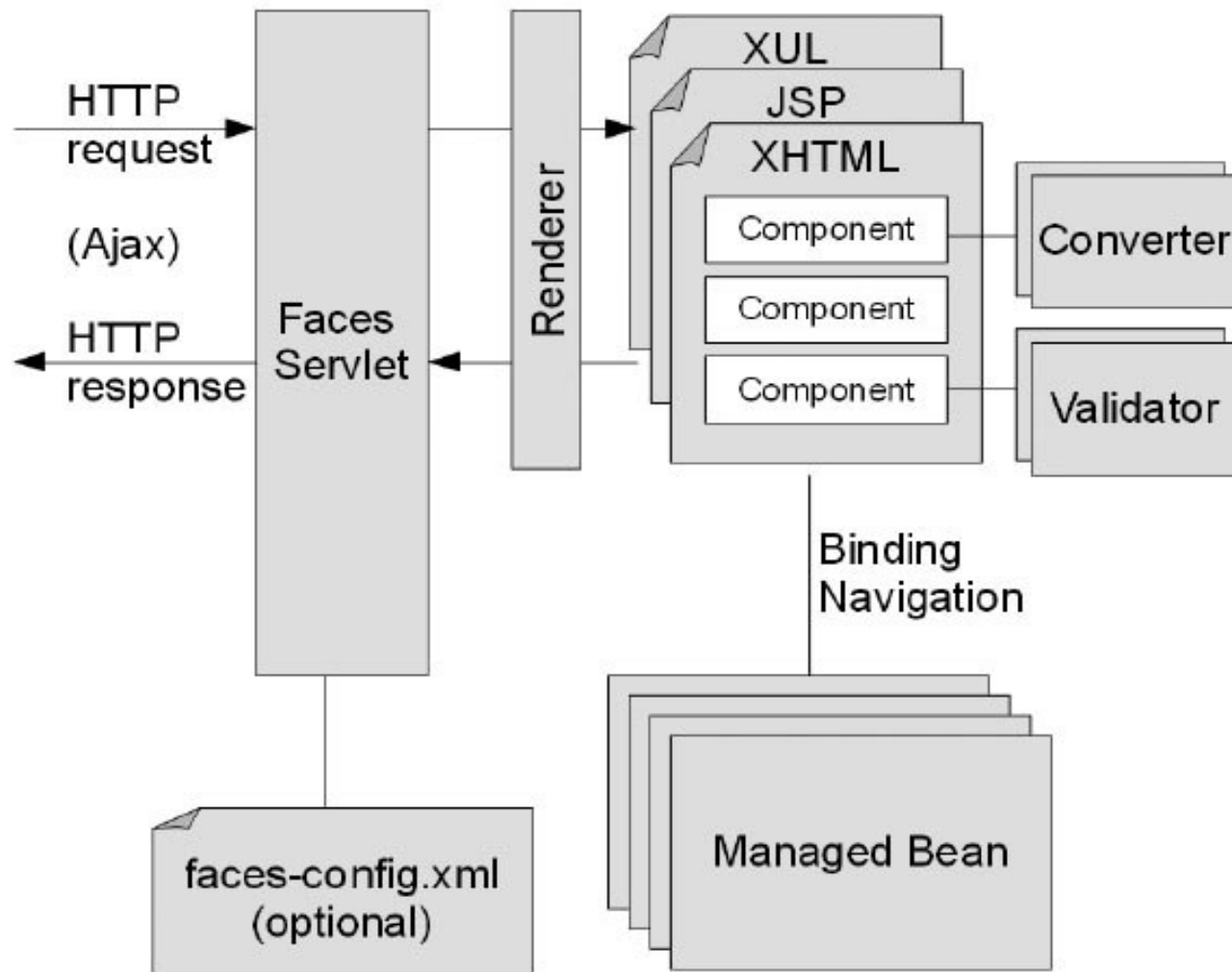
```
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:h="http://xmlns.jcp.org/jsf/html"  
      xmlns:f="http://xmlns.jcp.org/jsf/core">
```

JSF HTML tag reference

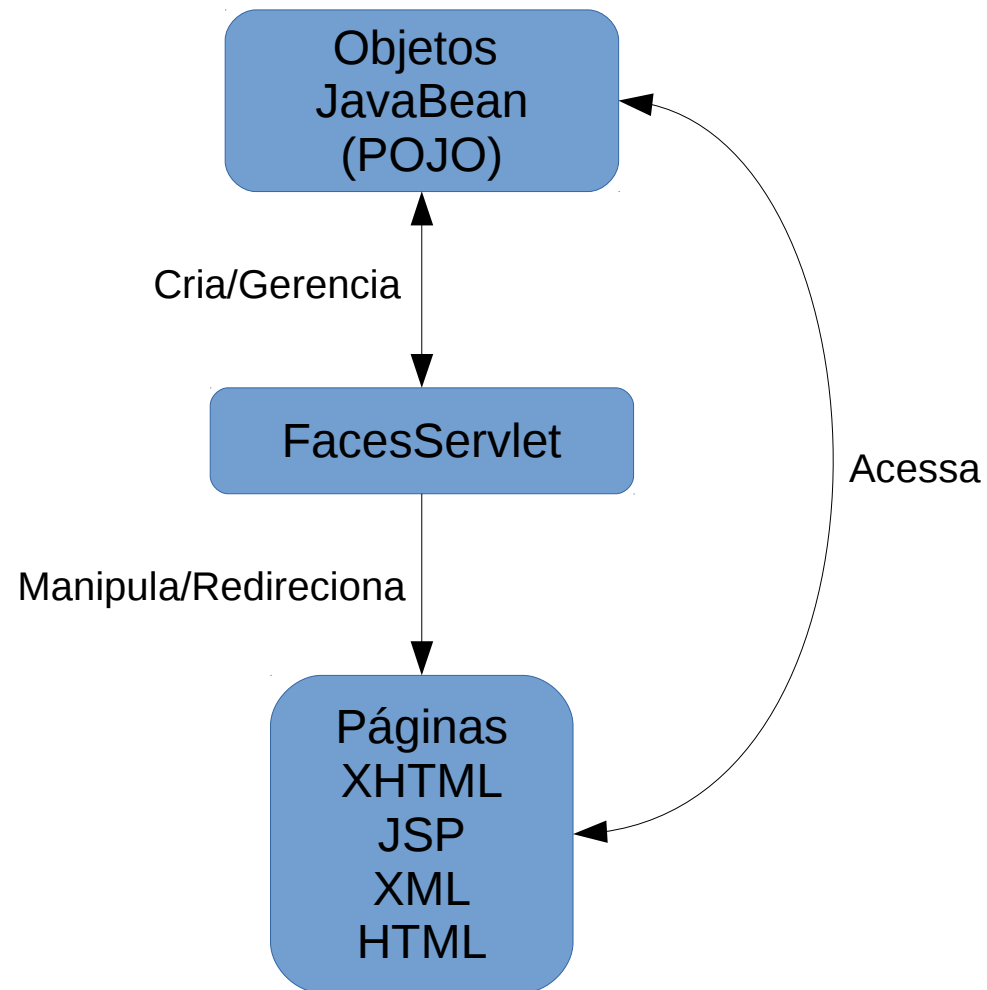
JSF Core tag reference

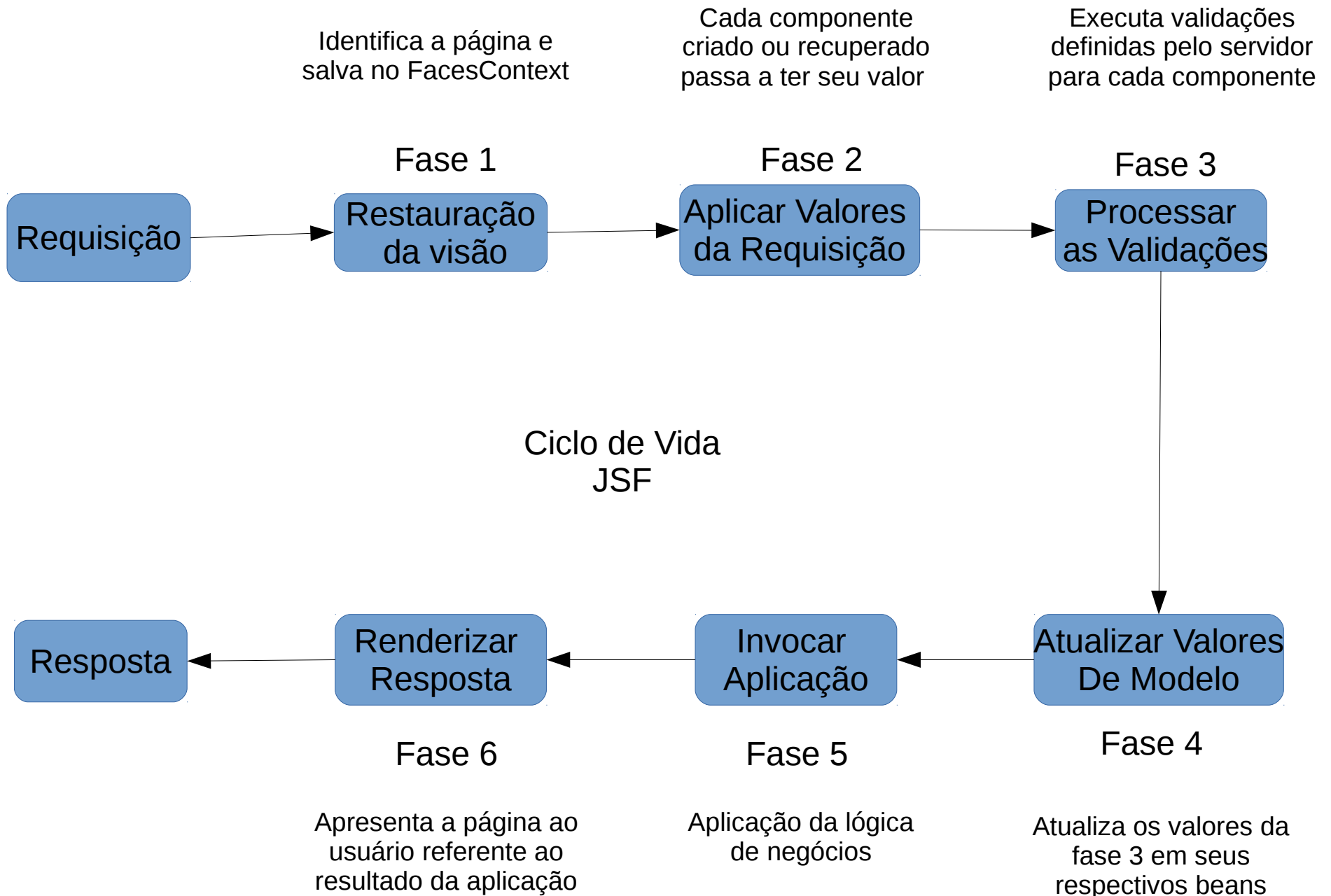


# Arquitetura

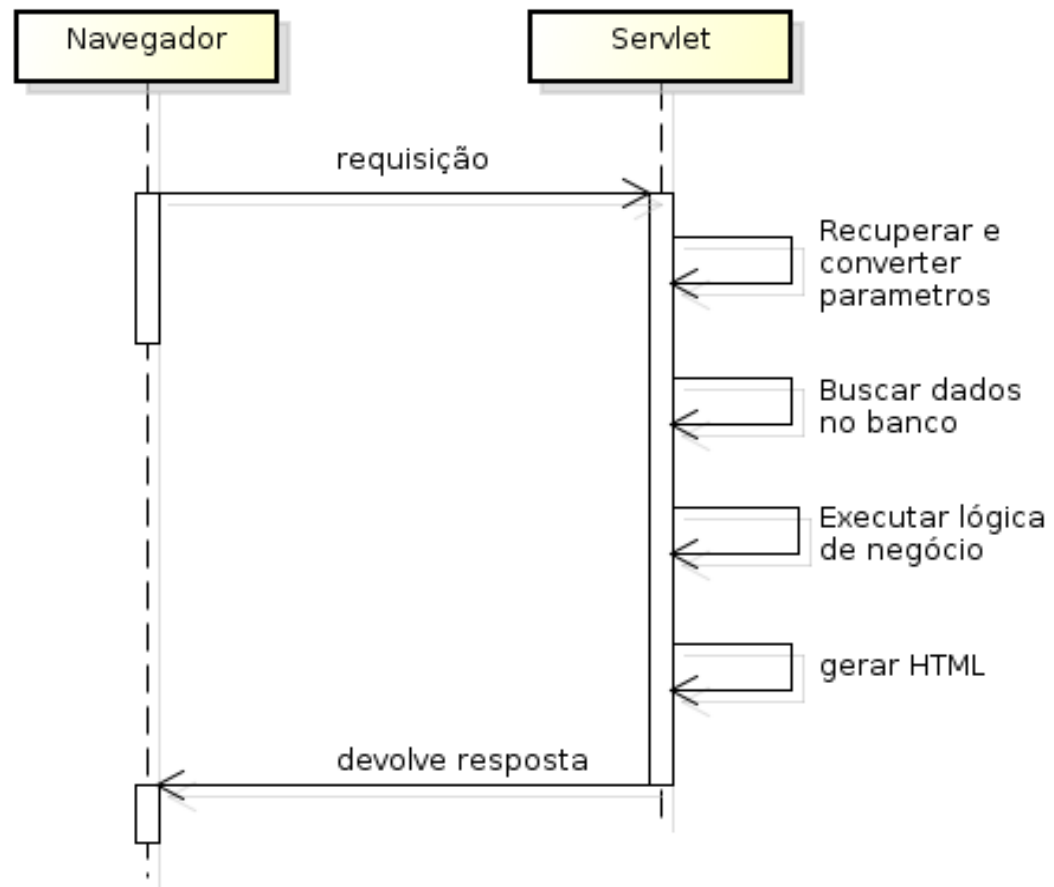


# MVC

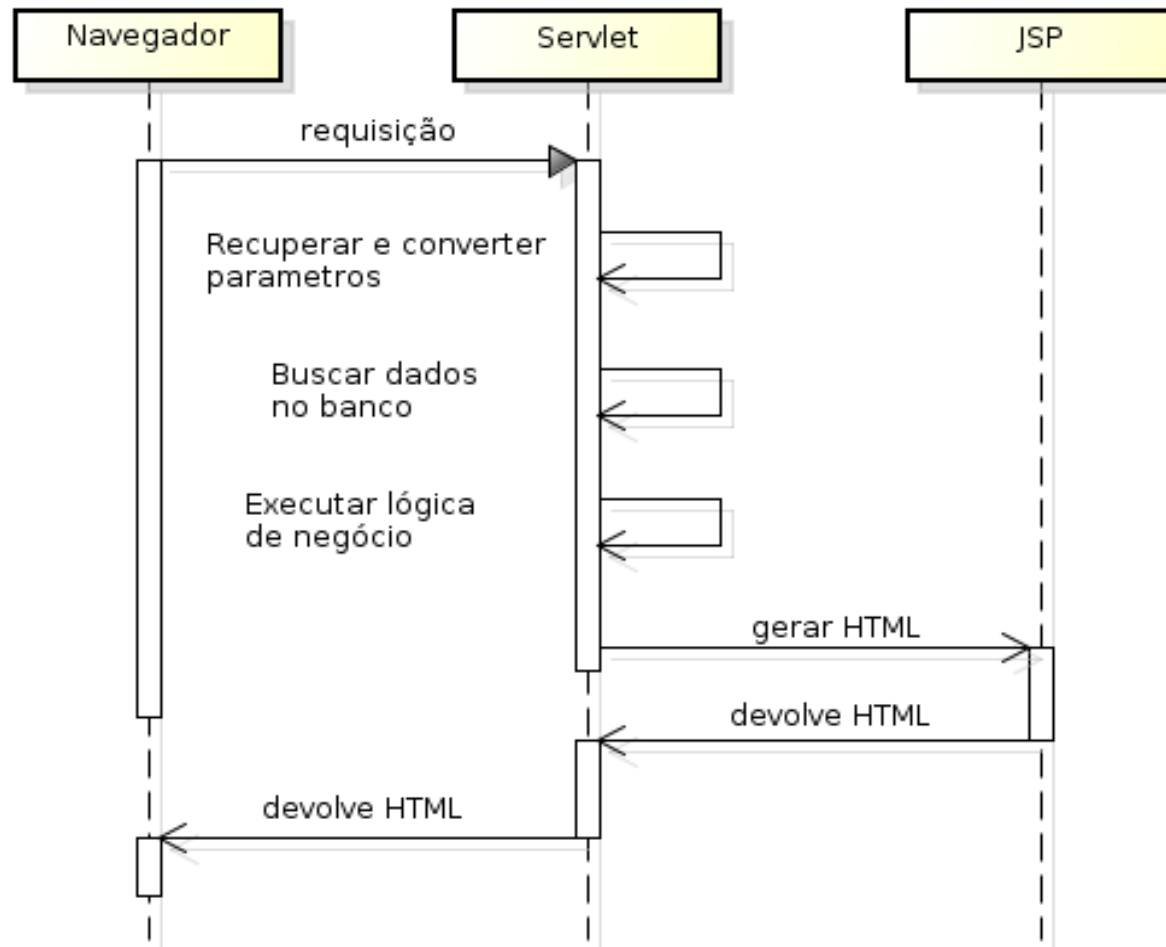




## Fluxo com tudo no Servlet

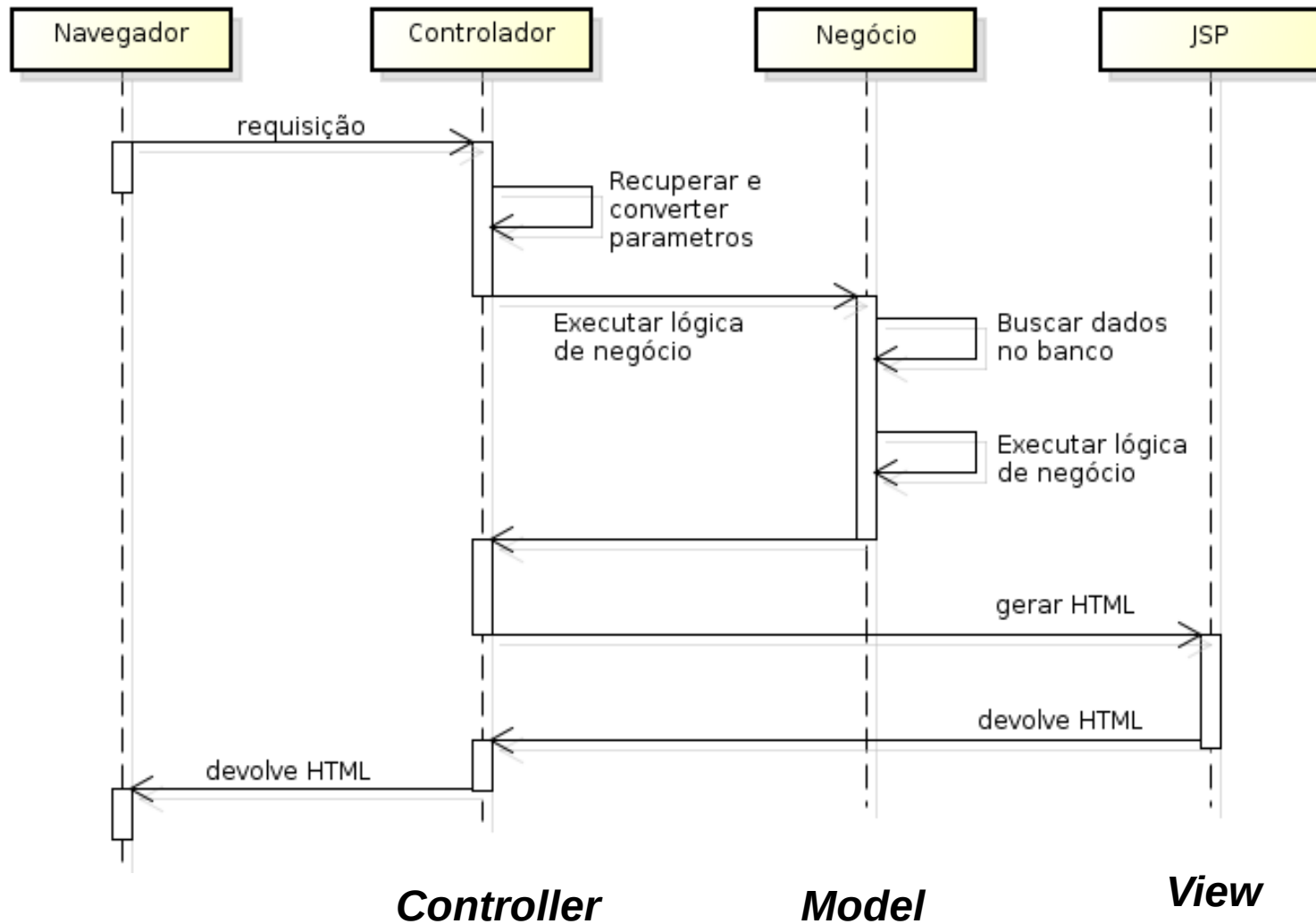


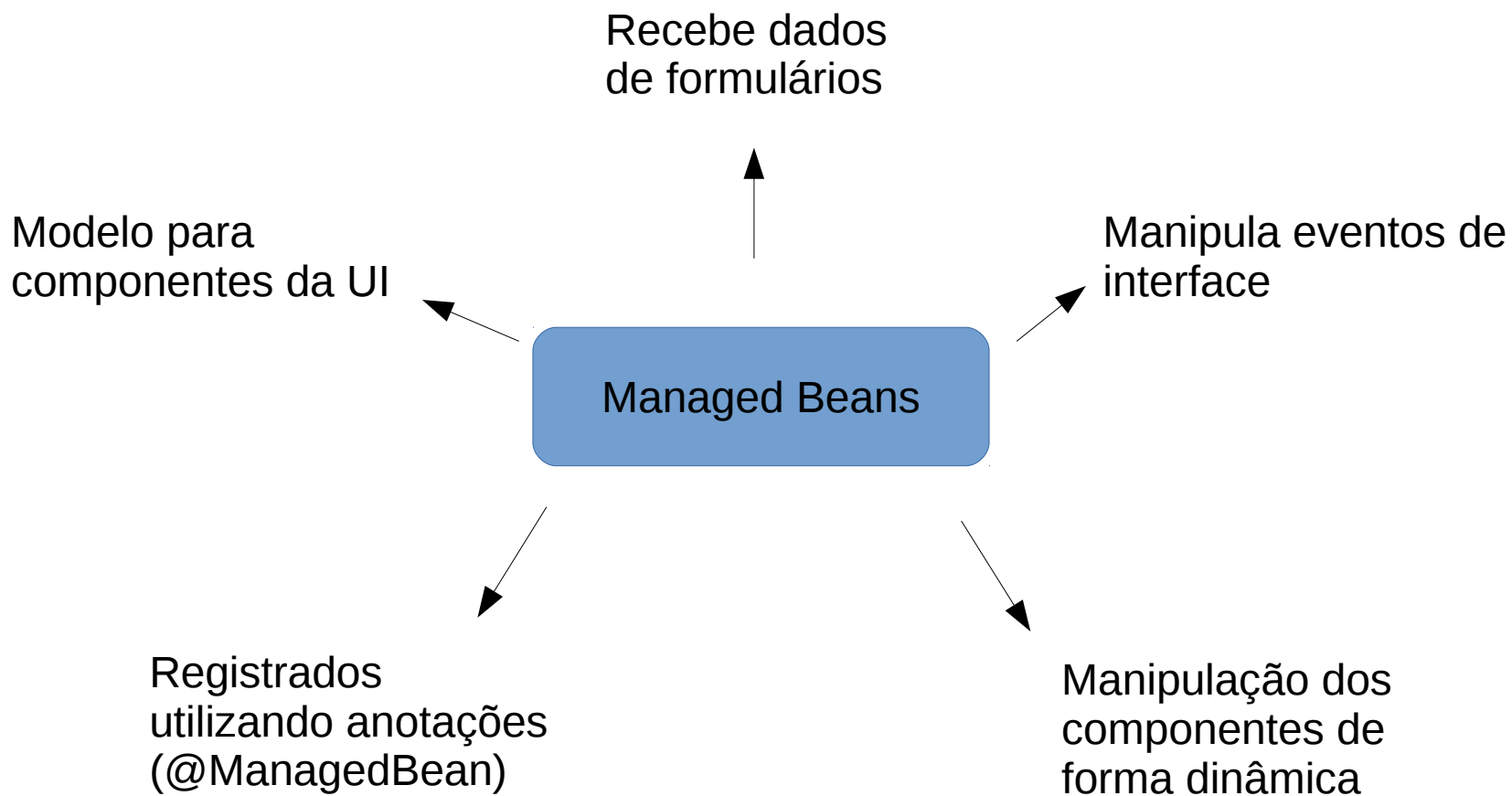
## Fluxo com geração do HTML fora do Servlet





## Fluxo com tarefas bem distribuídas





**SessionScoped ( @SessionScoped )**

**ViewScoped ( @ViewScoped )**

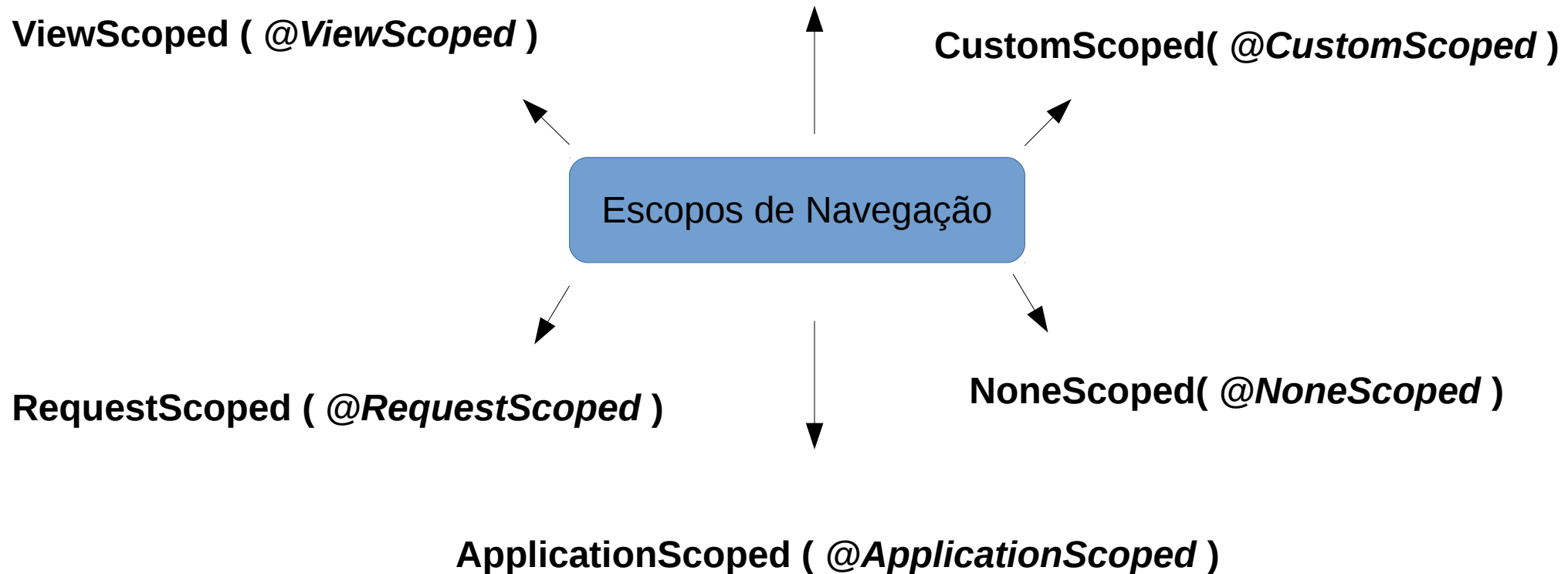
**CustomScoped( @CustomScoped )**

Escopos de Navegação

**RequestScoped ( @RequestScoped )**

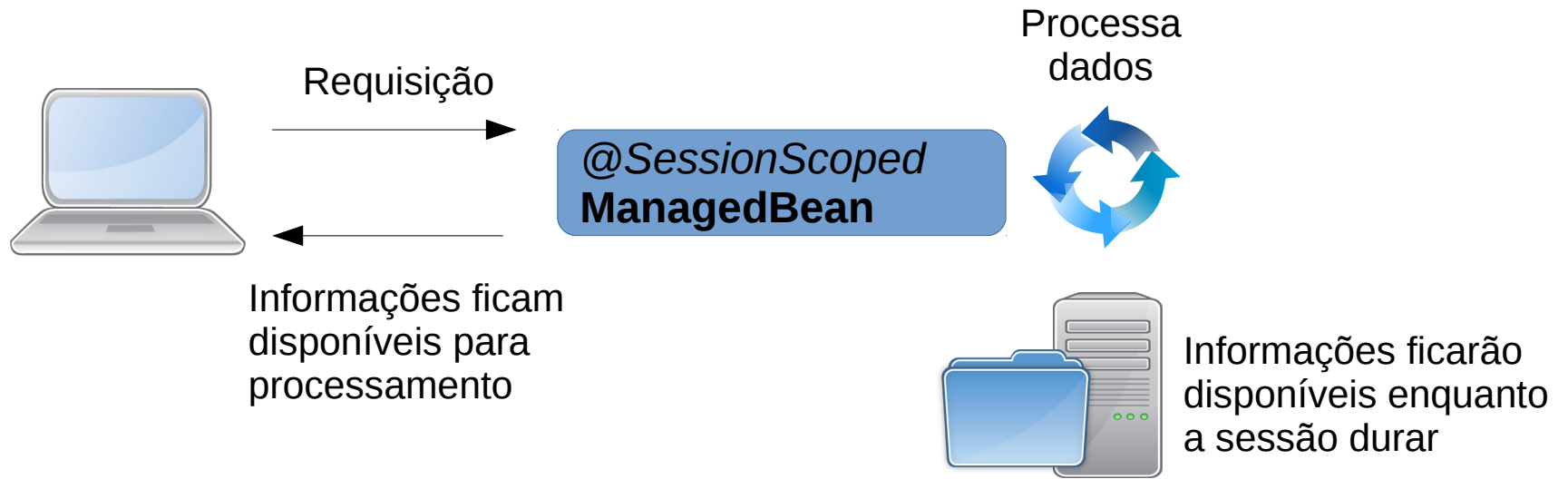
**NoneScoped( @NoneScoped )**

**ApplicationScoped ( @ApplicationScoped )**

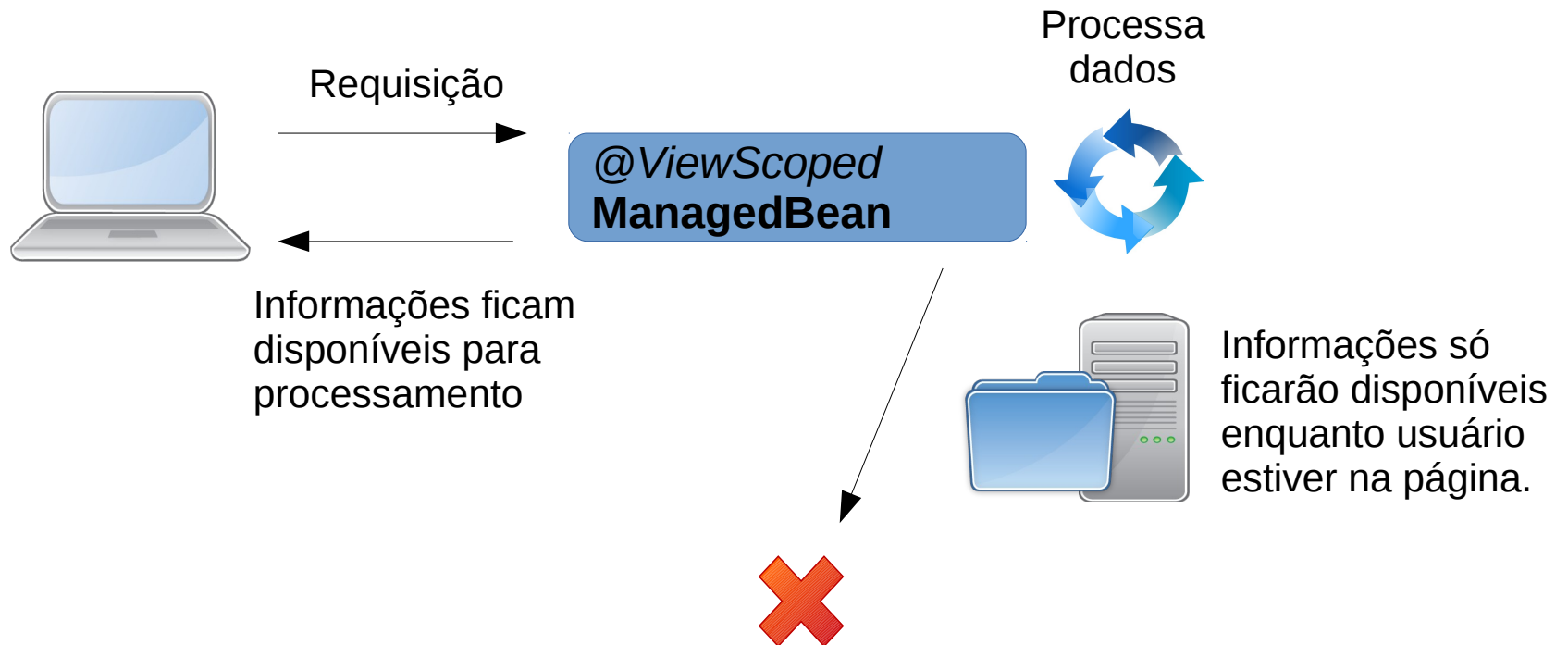


## SessionScoped ( @SessionScoped )

Todo atributo de um ManagedBean SessionScoped terá seu valor mantidos até o fim da sessão do usuário.

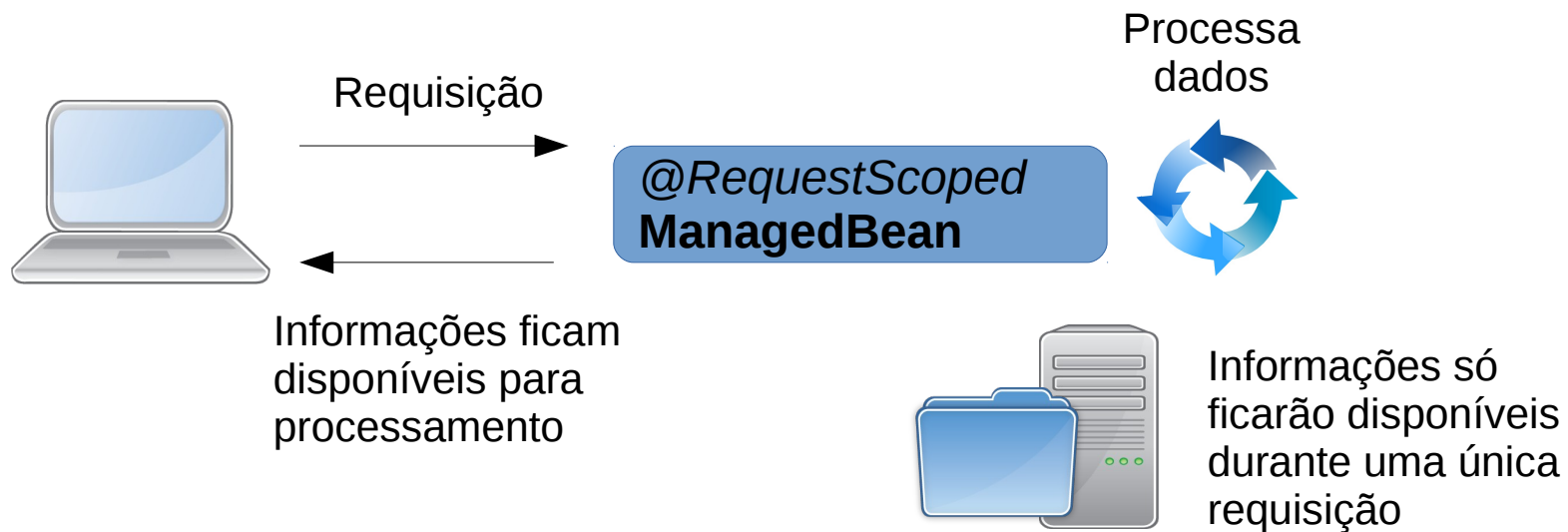


**ViewScoped ( @ViewScoped )**  
Existe na memória enquanto  
o usuário permanecer na página exibida.



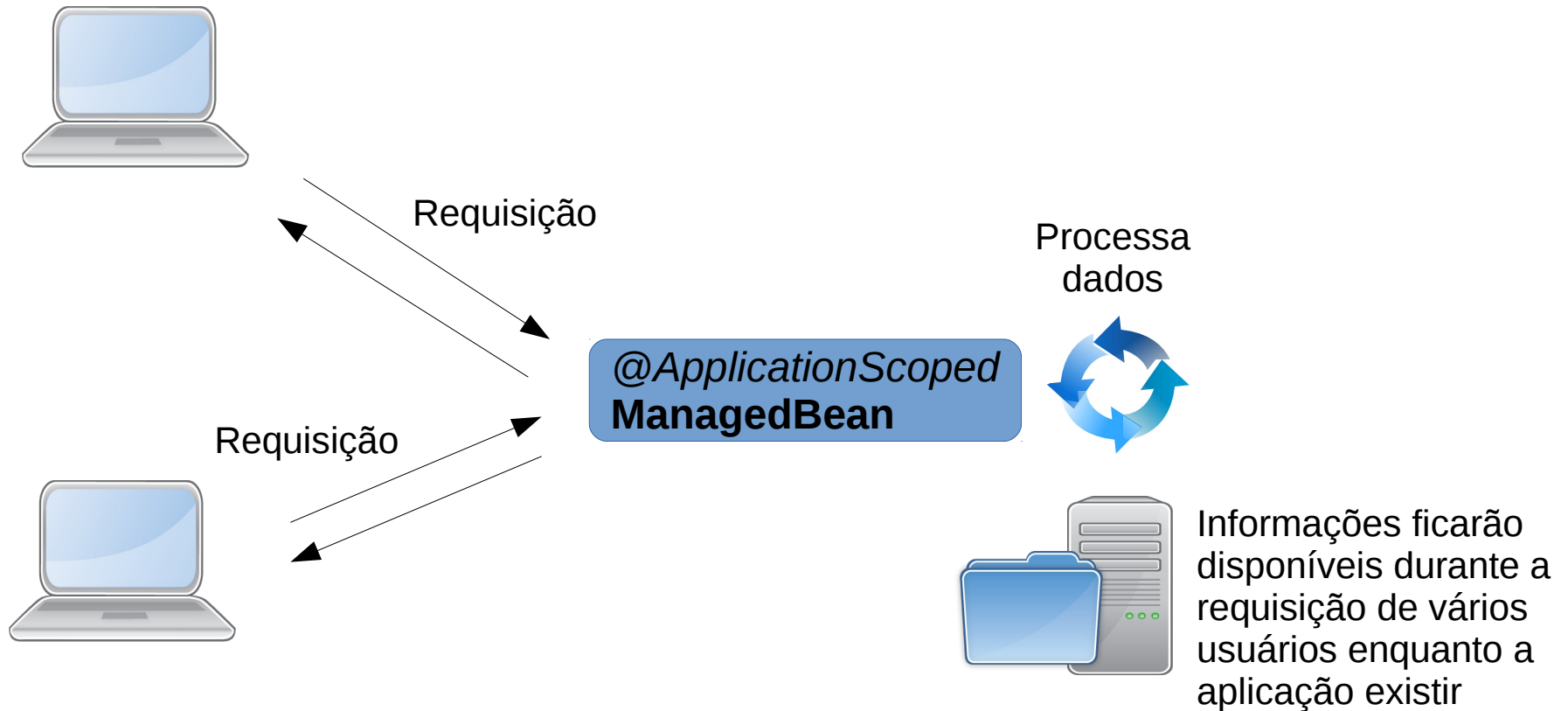
## RequestScoped ( @RequestScoped )

O Managed-Bean não manterá seu estado entre as chamadas do usuário.



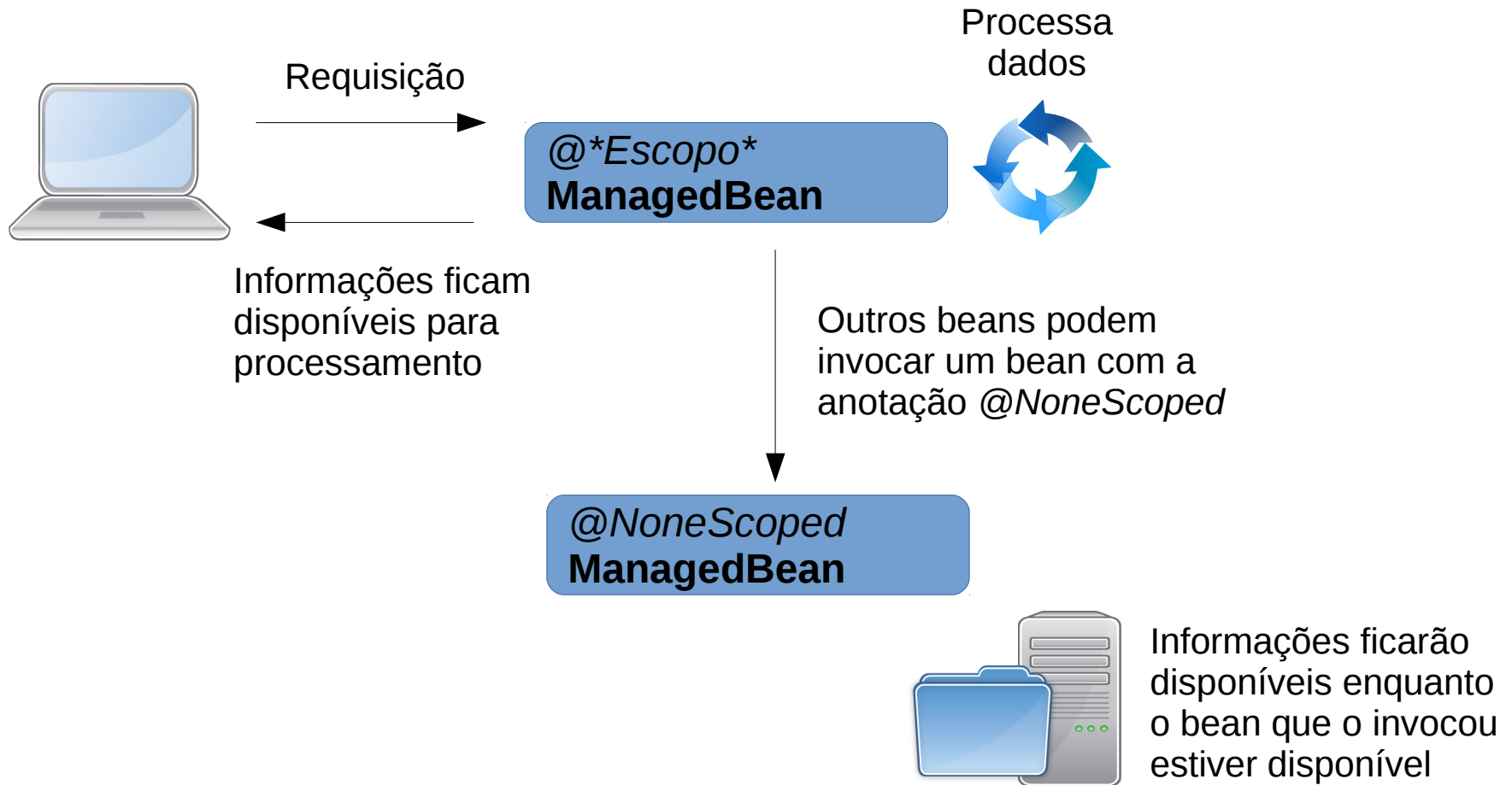
## **ApplicationScoped ( @ApplicationScoped )**

Existe enquanto a aplicação estiver sendo executada.



## NoneScoped( @NoneScoped )

O Managed Bean será invocado por outros beans e o seu tempo de vida vai ser de acordo com o escopo de quem o invocou





# Managed Bean

Escopo

Nome do  
ManagedBean

```
@ViewScoped
@ManagedBean(name="helloWorldBean")
public class HelloWorldBean implements Serializable {

    private static final long serialVersionUID = 6949827676782977015L;
    private String nome;

    public String getNome(){
        return nome;
    }

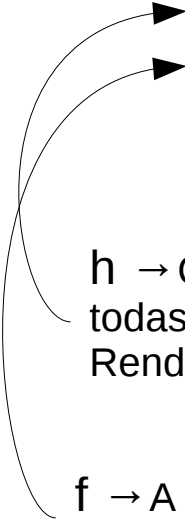
    public void setNome(String nome){
        this.nome = nome;
    }

}
```

Ainda poderíamos ter  
alguma lógica de  
negócios em nossos  
Managed Beans

```
<h:inputText value="#{helloWorldBean.nome}"/>
```

# Componentes UI



```
xmlns:h="http://xmlns.jcp.org/jsf/html"  
xmlns:f="http://xmlns.jcp.org/jsf/core"
```

h → Contém tags de componentes JavaServer Faces para todas as combinações UIComponent + HTML RenderKit Renderer definidas na especificação JavaServer Faces

f → A JSF Core tag library contém tags para fins de conversão, validação, Ajax e alguns outros usos.

Componentes UI podem ser estilizados por CSS e também interagem com *Managed Beans*

# Exemplo de utilização de algumas tags

## Página xhtml de exemplo

```
<h:head>
<title>JSF 2 Hello World</title>
</h:head>

<h:body>
<h3>JSF 2 Hello World Exemplo - hello.xhtml</h3>
<h:form id="form">
  <h:inputText value="#{helloWorldBean.nome}"/>
  <h:commandButton value="Bem Vindo">
    <f:ajax execute="form" render="form" />
  </h:commandButton>
  <br/><br/>
  <h:outputLabel rendered="#{helloWorldBean.nome != isEmpty}" value="Bem Vindo #{helloWorldBean.nome}"/>
</h:form>
</h:body>

</html>
```

## Menu de seleção única

```
<h:selectOneMenu id="list">
  <f:selectItem itemLabel="Opção 1" itemValue="1" />
  <f:selectItem itemLabel="Opção 2" itemValue="2" />
</h:selectOneMenu>
```

## Exemplo de utilização de algumas tags

### Conversor

```
<h:outputText id="balance" value="#{accountBean.balance}">
    <f:convertNumber currencySymbol="$" groupingUsed="true"
        maxFractionDigits="2" type="currency" />
</h:outputText>
```

```
<h:inputText id="nascimento">
    <f:convertDateTime type="date" pattern="dd/MM/yyyy" timeZone="America/Sao_Paulo"/>
</h:inputText>
```

### Validador

```
<h:inputSecret id="senha" value="#{validadoresBean.atributoObrigatorio}" required="true">
    <f:validateLength minimum="5"/>
</h:inputSecret>
```

```
<h:inputText id="withdraw" value="#{accountBean.withdraw.amount}">
    <f:validateDoubleRange minimum="20.00" maximum="1000.00" />
</h:inputText>
```

## Exemplo de utilização de algumas tags

### Lista de seleção única

```
private ClienteTable[] clienteList = new ClienteTable[] {  
    new ClienteTable("Cliente 01", 1),  
    new ClienteTable("Cliente 02", 2),  
    new ClienteTable("Cliente 03", 3)  
};  
  
private Map<String, Integer> clienteMap = new HashMap<String, Integer>();  
  
public Map<String, Integer> criarMap() {  
    clienteMap.put("Cliente 01", 1);  
    clienteMap.put("Cliente 02", 2);  
    clienteMap.put("Cliente 03", 3);  
  
    return clienteMap;  
}
```

## Exemplo de utilização de algumas tags

### Lista de seleção única

```
<!-- Lista criada na mão -->
<h:selectOneListbox id="Opcoes">
    <f:selectItem id="opcao1" itemLabel="Opção 1" itemValue="1"/>
    <f:selectItem id="opcao2" itemLabel="Opção 2" itemValue="2"/>
    <f:selectItem id="opcao3" itemLabel="Opção 3" itemValue="3"/>
</h:selectOneListbox>

<!-- Lista criada usando um array de objetos -->
<h:selectOneListbox id="Opcoes" value="Lista usando Array">
    <f:selectItems var="cliente" itemLabel="#{cliente.nome}"
        itemValue="#{cliente.codigo}" value="#{clienteData.clienteList}"/>
</h:selectOneListbox>

<!-- Lista criada usando um HashMap -->
<h:selectOneListbox id="Opcoes" value="Lista usando HashMap">
    <f:selectItems value="#{clienteData.criarMap()}" />
</h:selectOneListbox>
```

Podemos utilizar o atributo *value* passando como parâmetro uma array de objetos ou um Map

### Lista de seleção Unica

Opção 1	Cliente 01	Cliente 03
Opção 2	Cliente 02	Cliente 02
Opção 3	Cliente 03	Cliente 01

# Tabela de dados

Passamos uma lista de clientes para a *dataTable*

```
@ManagedBean(name = "clienteData")
@SessionScoped
public class ClienteTableData implements Serializable {

    private static final long serialVersionUID = 1L;

    private ClienteTable[] clienteList = new ClienteTable[] {
        new ClienteTable("Cliente 01", 1),
        new ClienteTable("Cliente 02", 2)
    };

    public ClienteTable[] getClienteList() {

        return clienteList;
    }
}
```

```
<h:body>
  <h:dataTable id="table1" value="#{clienteData.clienteList}"
    var="clienteTable" border="1">
    <f:facet name="header">
      <h:outputText value="Lista de Clientes" />
    </f:facet>
    <h:column>
      <f:facet name="header">
        <h:outputText value="Cliente" />
      </f:facet>
      <h:outputText value="#{clienteTable.nome}" />
    </h:column>
    <h:column>
      <f:facet name="header">
        <h:outputText value="Código" />
      </f:facet>
      <h:outputText value="#{clienteTable.codigo}" />
    </h:column>
  </h:dataTable>
</h:body>
```

Lista de Clientes	
Cliente	Código
Cliente 01	1
Cliente 02	2

Criamos uma coluna para cada atributo que desejamos mostrar e utilizamos um *outputText* para acessar o atributo no bean.

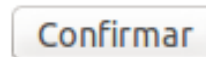
# Command Link e Command Button

## <h:commandButton>

Gera um *submit button* que pode ser associado a um bean ou a uma classe ActionListener para lidar com eventos.

```
<h:form>
  Command Button
  <h:commandButton value="Confirmar" action="" />
  Command Link
  <h:commandLink value="Redirecionar" action="" />
</h:form>
```

Command Button



Command Link  
[Redirecionar](#)

## <h:commandLink>

Gera uma tag de link <a> que se comporta como um *submit button* e que pode ser associada a um bean ou uma classe ActionListener para lidar com eventos. Utiliza Java Script para enviar o formulário.



# Ajax com HTML e Java Script

```
<body>
  <button type="button" onclick="acionarAjax()">Ajax!</button>
  <p id="output" />

  <script type="text/javascript">
    function acionarAjax() {

      var xhttp = new XMLHttpRequest();
      xhttp.onreadystatechange = function() {
        if (xhttp.readyState == 4 && xhttp.status == 200) {
          document.getElementById("output").innerHTML = xhttp.responseText;
        }
      };

      xhttp.open('GET', "texto.txt", true);
      xhttp.send();
    }
  </script>
</body>
```

Elemento com id  
"output" irá receber a  
resposta.

Método da requisição HTTP, URL e  
se a chamada será assíncrona.

Envia qualquer dado adicional caso a  
requisição seja um post.

# Ajax com JSF

**execute="form"** indica que o formulário com esse ID será enviado para o servidor para ser processado.

**render="output"** indica que depois da requisição Ajax, o componente com esse ID será atualizado.

```
<h:form id="form">
  <h:inputText value="#{helloWorldBean.nome}"/>
  <h:commandButton value="Bem Vindo">
    <f:ajax execute="form" render="form" />
  </h:commandButton>
  <br/><br/>
  <h:outputLabel rendered="#{helloWorldBean.nome != isEmpty}" value="Bem Vindo #{helloWorldBean.nome}"/>
</h:form>
```

## JSF 2 Hello World Exemplo - hello.xhtml

3Way Bem Vindo

Bem Vindo 3Way

Botão aciona a requisição Ajax

## Conversor Padrão

```
<h:inputText id="nascimento">  
  <f:convertDateTime type="date" pattern="dd/MM/yyyy" timeZone="America/Sao_Paulo"/>  
</h:inputText>
```

converte entradas do usuário em valores do tipo ***java.util.Date*** com um formato padrão.

```
<h:inputText id="peso">  
  <f:convertNumber integerOnly="true"/>  
</h:inputText>
```

Converte entradas do usuário em valores do tipo ***java.lang.Number***.

# Conversor Customizado

```
@FacesConverter("celsiusToFahrenheitConverter")  
public class CelsiusToFahrenheitConverter implements Converter{
```

Anotação indicando que a classe é um converter + nome do converter

Interface Converter

Método que recebe a string e converte para objeto

```
@Override  
public Object getAsObject(FacesContext context, UIComponent component, String value) {  
    Float resultado = 0F;  
    try {  
        Float celsius = Float.parseFloat(value);  
        resultado = (celsius * 9/5) + 32;  
    } catch (Exception e) {  
        FacesMessage msg = new FacesMessage  
            ("Erro de conversão em celsiusToFahrenheitConverter", "Entrada inválida, tente novamente.");  
        msg.setSeverity(FacesMessage.SEVERITY_ERROR);  
  
        throw new ConverterException(msg);  
    }  
    return resultado;  
}
```

Objeto do tipo Float

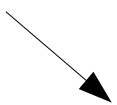
```
@Override  
public String getAsString(FacesContext context, UIComponent component, Object value) {  
    return value.toString();  
}
```

Método que recebe o objeto e converte para String

```
<h:inputText id="celsius" value="#{conversoresBean.celsiusToFahrenheit}">  
    <f:converter converterId="celsiusToFahrenheitConverter"/>  
</h:inputText>
```

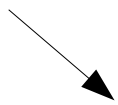
## Validador Padrão

```
<h:inputSecret id="senha" value="#{validadoresBean.atributoObrigatorio}" required="true">  
  <f:validateLength minimum="5"/>  
</h:inputSecret>
```



Verifica o comprimento de um valor e o limita no intervalo especificado.

```
<h:inputText id="withdraw" value="#{accountBean.withdraw.amount}">  
  <f:validateDoubleRange minimum="20.00" maximum="1000.00" />  
</h:inputText>
```



Valida entradas do usuário quando a entrada esperada for do tipo ponto flutuante (float).

## Validador Customizado

Anotação indicando que a classe é um validator + nome do validator

```
@FacesValidator("emailValidator")  
public class EmailValidator implements Validator {
```

Interface Validator

```
    private static final String EMAIL_PATTERN =  
        "^[_A-Za-z0-9-]+(\\\\" + "[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\\\\" + "[A-Za-z0-9-]+)*" + "(\\\\" + "[A-Za-z]{2,})$";  
    private Pattern pattern;  
    private Matcher matcher;  
  
    public EmailValidator(){  
        pattern = Pattern.compile(EMAIL_PATTERN);  
    }  
  
    @Override  
    public void validate(FacesContext context, UIComponent component, Object value) throws ValidatorException {  
        matcher = pattern.matcher(value.toString());  
        if(!matcher.matches()){  
            FacesMessage msg = new FacesMessage("Validação de email falhou", "Email informao inválido");  
            msg.setSeverity(FacesMessage.SEVERITY_ERROR);  
  
            throw new ValidatorException(msg);  
        }  
    }  
}
```

```
<h:inputText id="email" value="#{validadoresBean.email}" required="true">  
    <f:validator validatorId="emailValidator"/>  
</h:inputText>
```

# Mensagens

Mensagem será exibida caso exceção seja lançada.

```
FacesMessage msg = new FacesMessage  
    ("Erro de conversão em celsiusToFahrenheitConverter", "Entrada inválida, tente novamente.");  
msg.setSeverity(FacesMessage.SEVERITY_ERROR);  
throw new ConverterException(msg);
```

```
FacesMessage msg = new FacesMessage("Validação de email falhou", "Email informao inválido");  
msg.setSeverity(FacesMessage.SEVERITY_ERROR);  
throw new ValidatorException(msg);
```

```
FacesMessage msg = new FacesMessage (severidade, mensagem, detalhes);  
FacesContext context = FacesContext.getCurrentInstance();  
Context.addMessage(client_id, msg);
```

```
FacesMessage.Severity_ERROR  
FacesMessage.Severity_FATAL  
FacesMessage.Severity_INFO  
FacesMessage.Severity_WARN
```

# Internacionalização

Arquivo *.properties*

*nomeDoArquivo\_abreviacao\_local.properties*

messages\_en\_US.properties

messages\_pt\_BR.properties

messages

```
1 msg.title = JSF 2 Internationalization
2 msg.boasVindas = Welcome to the extraordinary world of Java!
```

messages

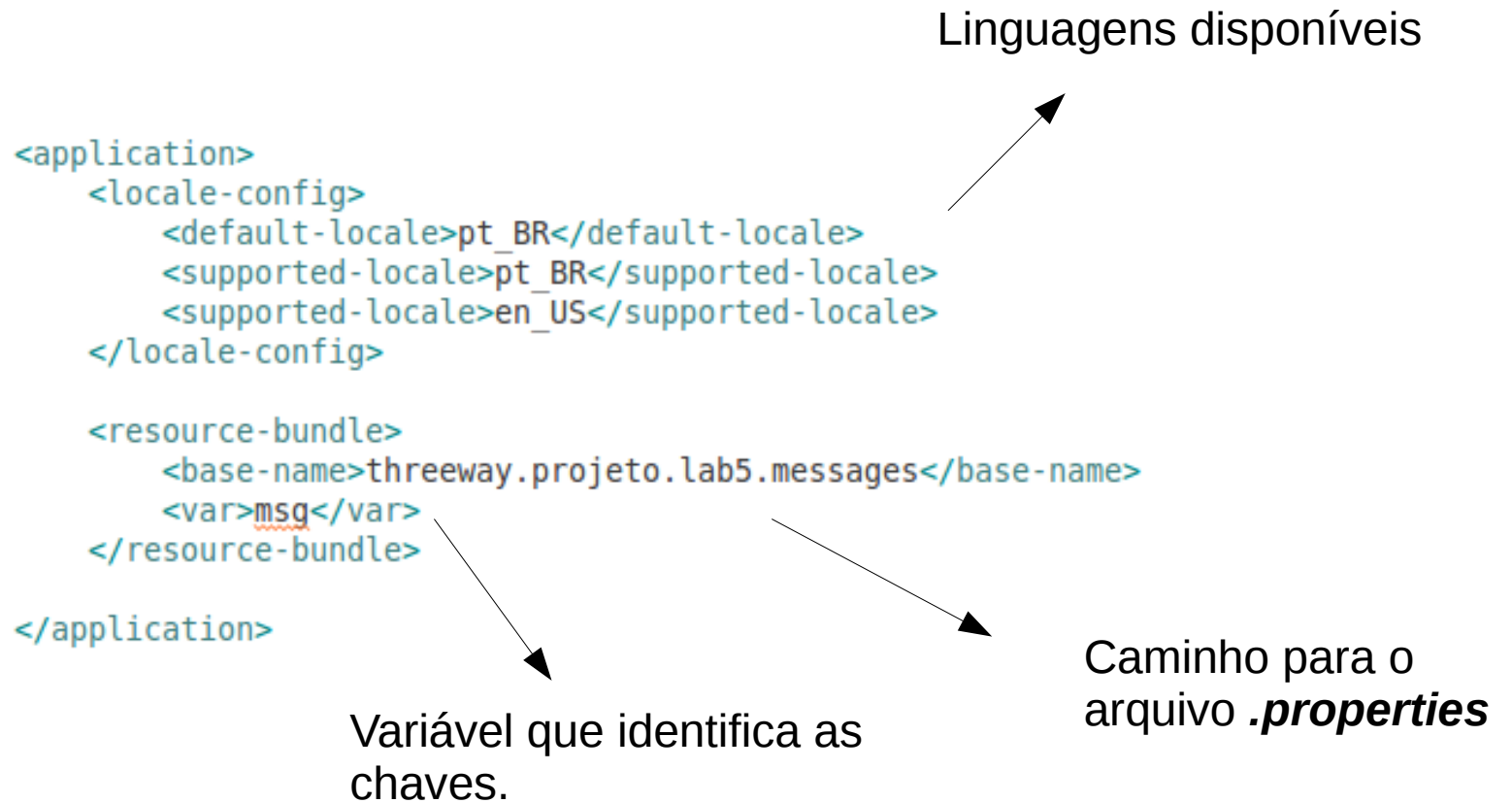
```
1 msg.title = JSF 2 Internacionalização
2 msg.boasVindas = Bem Vindo ao mundo extraordinário do Java!
```

**CHAVE = VALOR**

As mensagens são identificadas pelas chaves. Somente seus valores devem ser alterados.



## Mapeamento dos arquivos properties no ***faces-config.xml***



Navegação

```
graph LR; Navegação --> Implícita; Navegação --> Explícita; Implícita --> Estática1[Estática]; Implícita --> Dinâmica1[Dinâmica]; Explícita --> Estática2[Estática]; Explícita --> Dinâmica2[Dinâmica];
```

Implícita

- Estática
- Dinâmica

Explícita

- Estática
- Dinâmica

# Navegação Bean

Escopo deste  
Managed Bean

Nome do Managed Bean

```
@ViewScoped
@ManagedBean(name="navegacaoBean")
public class NavegacaoBean implements Serializable {

    private static final long serialVersionUID = -7622250974049755899L;

    public String navegacaoDinamicaImplicitaIndex() {
        return "index";
    }

    public String navegacaoDinamicaImplicitaPaginal() {
        return "paginal";
    }

    public String navegacaoDinamicaExplicitaIndex() {
        return "NavegacaoExplicitaIndex";
    }

    public String navegacaoDinamicaExplicitaPaginal() {
        return "navegacaoExplicitaPaginal";
    }

}
```

Métodos para realizar a  
navegação dinâmica

# faces-config.xml

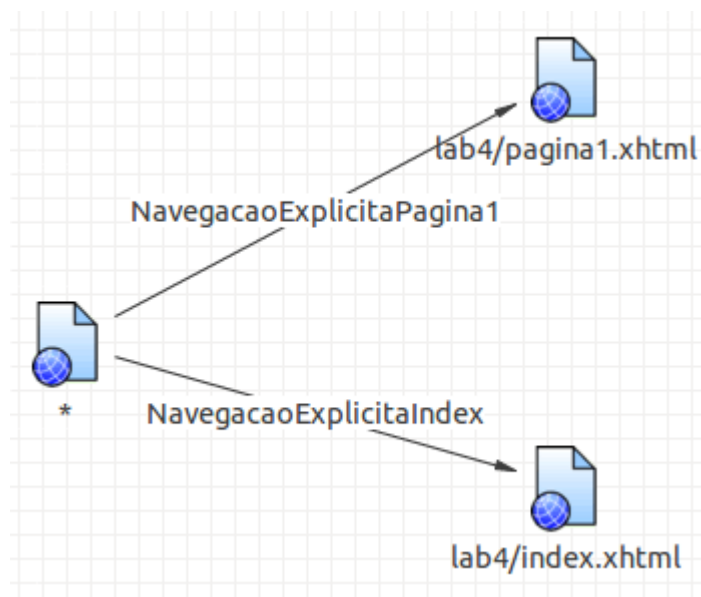
Configura uma regra de navegação entre as páginas.  
Ainda é possível inserir a tag `<from-view-id>` para determinar a página de origem da navegação.

```
<navigation-rule>  
  <navigation-case>  
    <from-outcome>NavegacaoExplicitaIndex</from-outcome>  
    <to-view-id>/lab4/index.xhtml</to-view-id>  
  </navigation-case>  
</navigation-rule>  
<navigation-rule>  
  <navigation-case>  
    <from-outcome>NavegacaoExplicitaPagina1</from-outcome>  
    <to-view-id>/lab4/pagina1.xhtml</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

Valor de saída do atributo  
*action* definido na UI

Página a ser exibida

Cada caso de navegação determina um caminho diferente partindo de uma mesma página.



# Navegação Explícita

```
<h:panelGrid cellpadding="2">
    Navegacao Estatica Explícita
    <h:commandButton value="Ir Para Index" action="NavegacaoExplícitaIndex"/>
    <h:commandButton value="Ir para Pagina 1" action="NavegacaoExplícitaPaginal"/>
</h:panelGrid>
<hr/>
<h:panelGrid cellpadding="2">
    Navegacao Dinamica Explícita
    <h:commandButton value="Ir Para Index"
        action="#{navegacaoBean.navegacaoDinamicaExplícitaIndex()}" />
    <h:commandButton value="Ir Para Pagina 1"
        action="#{navegacaoBean.navegacaoDinamicaExplícitaPaginal()}" />
</h:panelGrid>
```

Nome do *outcome* que define a página a ser exibida

Navegação é gerada dentro de um Managed Bean

```
public String navegacaoDinamicaExplícitaPaginal() {
    return "NavegacaoExplícitaPaginal";
}
```

Retorna o outcome que define a página a ser exibida.

```
public String navegacaoDinamicaExplícitaIndex() {
    return "NavegacaoExplícitaIndex";
}
```

# Navegação Implícita

```
<h:panelGrid cellpadding="2">
    Navegacao Estatica Implicita
    <h:commandButton value="Ir para index" action="index"/>
    <h:commandButton value="Ir para pagina 1" action="paginal"/>
</h:panelGrid>
<hr/>
<h:panelGrid cellpadding="2">
    Navegação Dinamica Implicita
    <h:commandButton value="Ir para Index"
        action="#{navegacaoBean.navegacaoDinamicaImplicitaIndex()}" />
    <h:commandButton value="Ir para Pagina 1"
        action="#{navegacaoBean.navegacaoDinamicaImplicitaPaginal()}" />
</h:panelGrid>
```

O nome da página a ser exibida é definido no próprio botão

A navegação é gerada dentro de um managed bean

```
public String navegacaoDinamicaImplicitaIndex() {
    return "index";
}
```

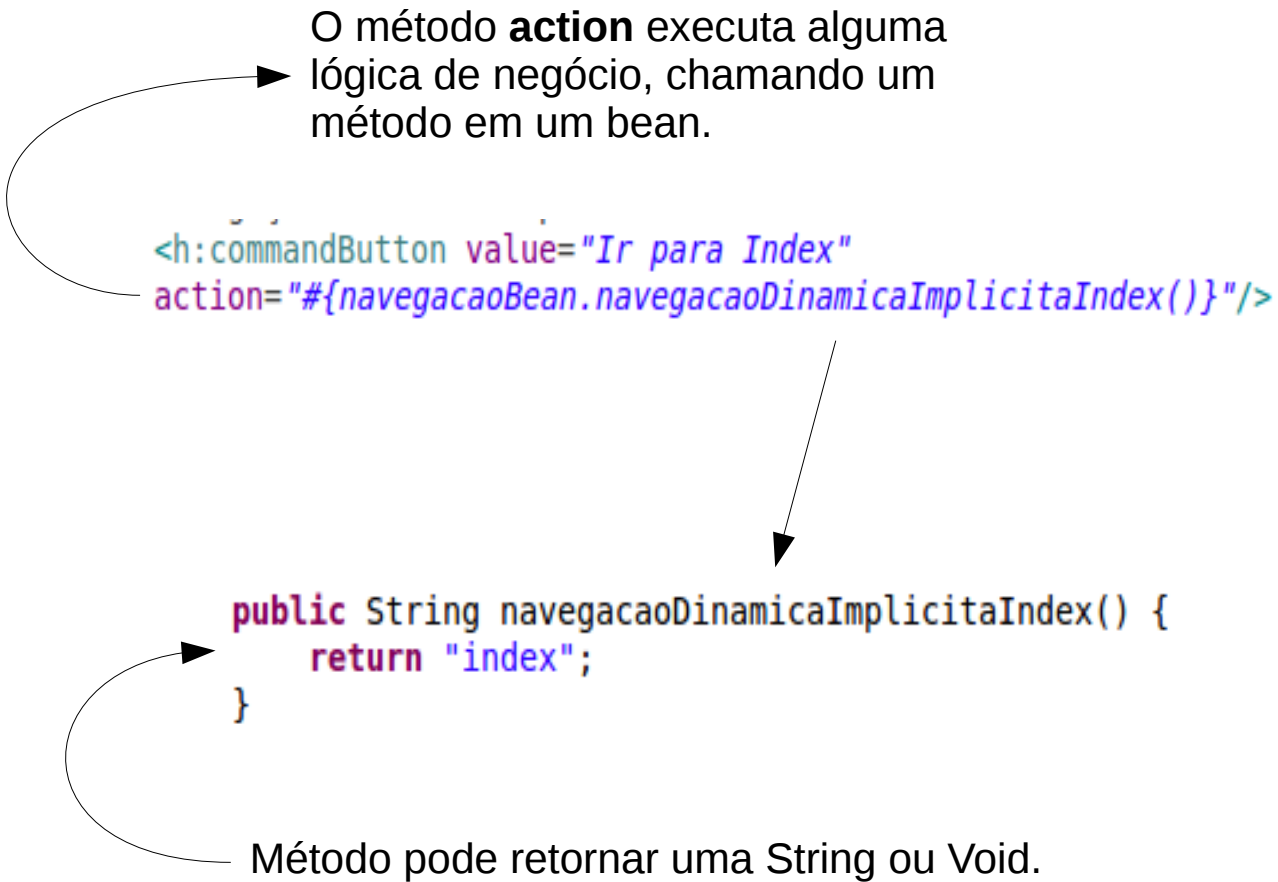
Retorna o nome da página a ser exibida.

```
public String navegacaoDinamicaImplicitaPaginal() {
    return "paginal";
}
```

## Action Handlers

Executam alguma ação e depois podem redirecionar o usuário para outra tela.

O método **action** executa alguma lógica de negócio, chamando um método em um bean.



```
<h:commandButton value="Ir para Index"
action="#{navegacaoBean.navegacaoDinamicaImplicitaIndex()}" />
```

The diagram illustrates the execution of an action handler. A curved arrow points from the `action` attribute in the HTML code to the `navegacaoDinamicaImplicitaIndex()` method in the Java code. A straight arrow points from the `navegacaoDinamicaImplicitaIndex()` method back to the `action` attribute, indicating the call sequence.

```
public String navegacaoDinamicaImplicitaIndex() {
    return "index";
}
```

Método pode retornar uma String ou Void.

# Event Listeners

Executam alguma ação e mantêm o usuário na mesma tela ou **podem** redirecioná-lo para outra tela **forçadamente**.

O **actionListener** Executa uma lógica relacionada a view.

```
<p:commandButton actionListener="#{manterClienteBean.limpar()}"  
update="panelCadastro :formNavegacao" value="#{msg['btn.cliente.limpar']}" />
```

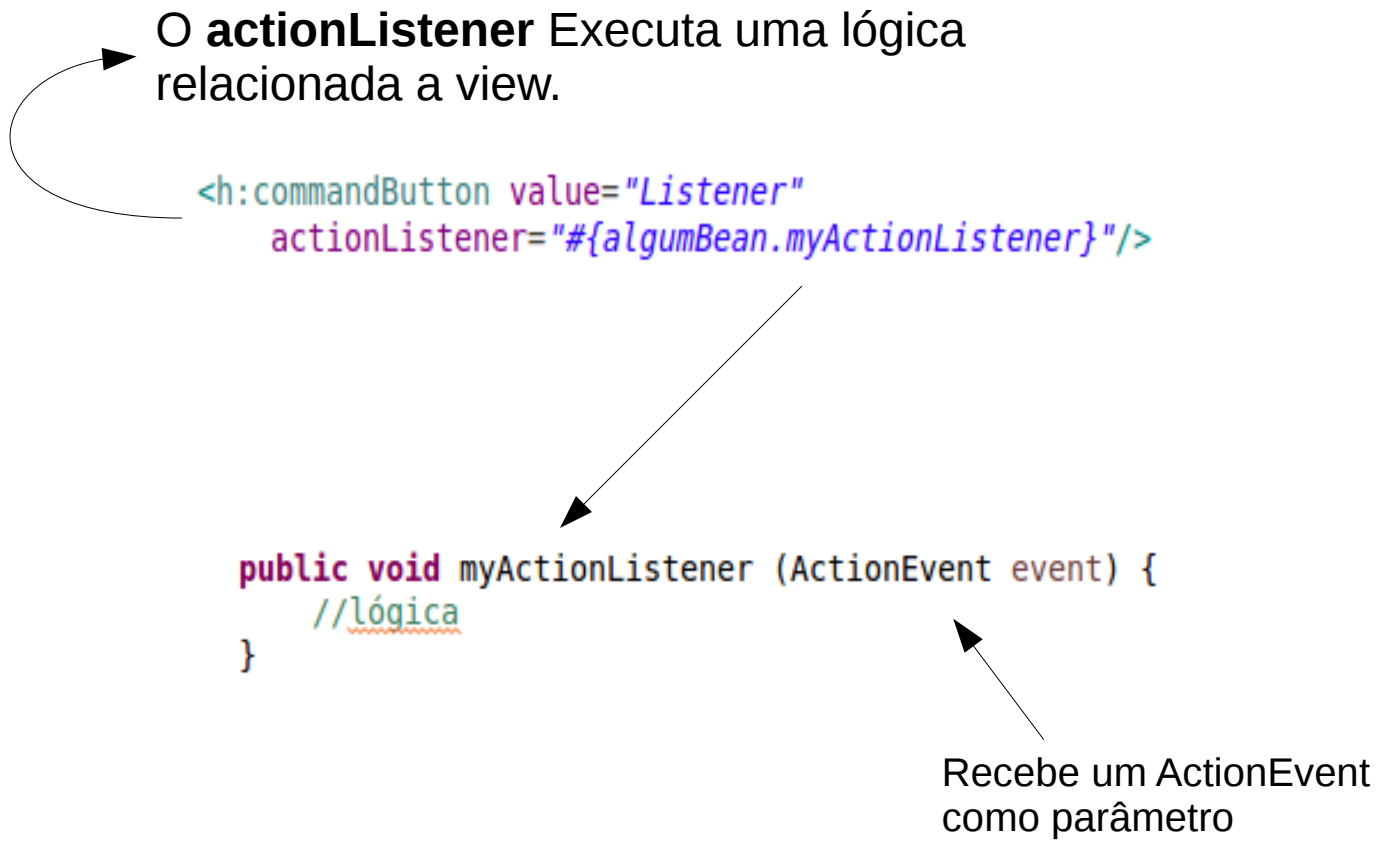
```
public void limpar() {  
    this.cliente = new Cliente();  
}
```

Método deve ter retorno Void.



# Event Listeners

O **actionListener** Executa uma lógica relacionada a view.



```
<h:commandButton value="Listener"
  actionListener="#{algumBean.myActionListener}"/>
```

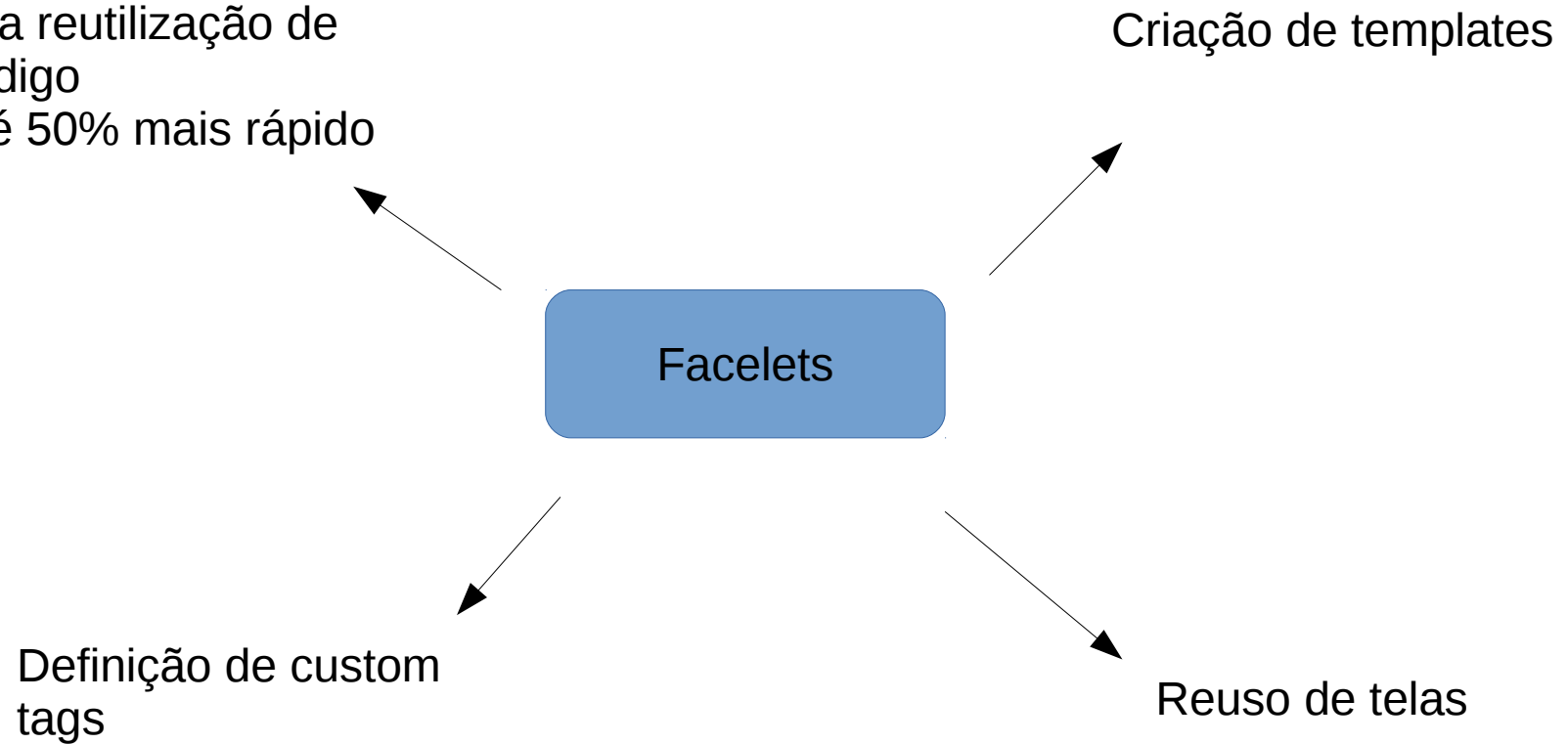
The diagram illustrates the flow of an event listener. A curved arrow points from the `actionListener` attribute in the JSF component to the `myActionListener` method. A straight arrow points from the `myActionListener` method to the `event` parameter in its signature. Another straight arrow points from the explanatory text 'Recebe um ActionEvent como parâmetro' to the `event` parameter.

```
public void myActionListener (ActionEvent event) {
    //lógica
}
```

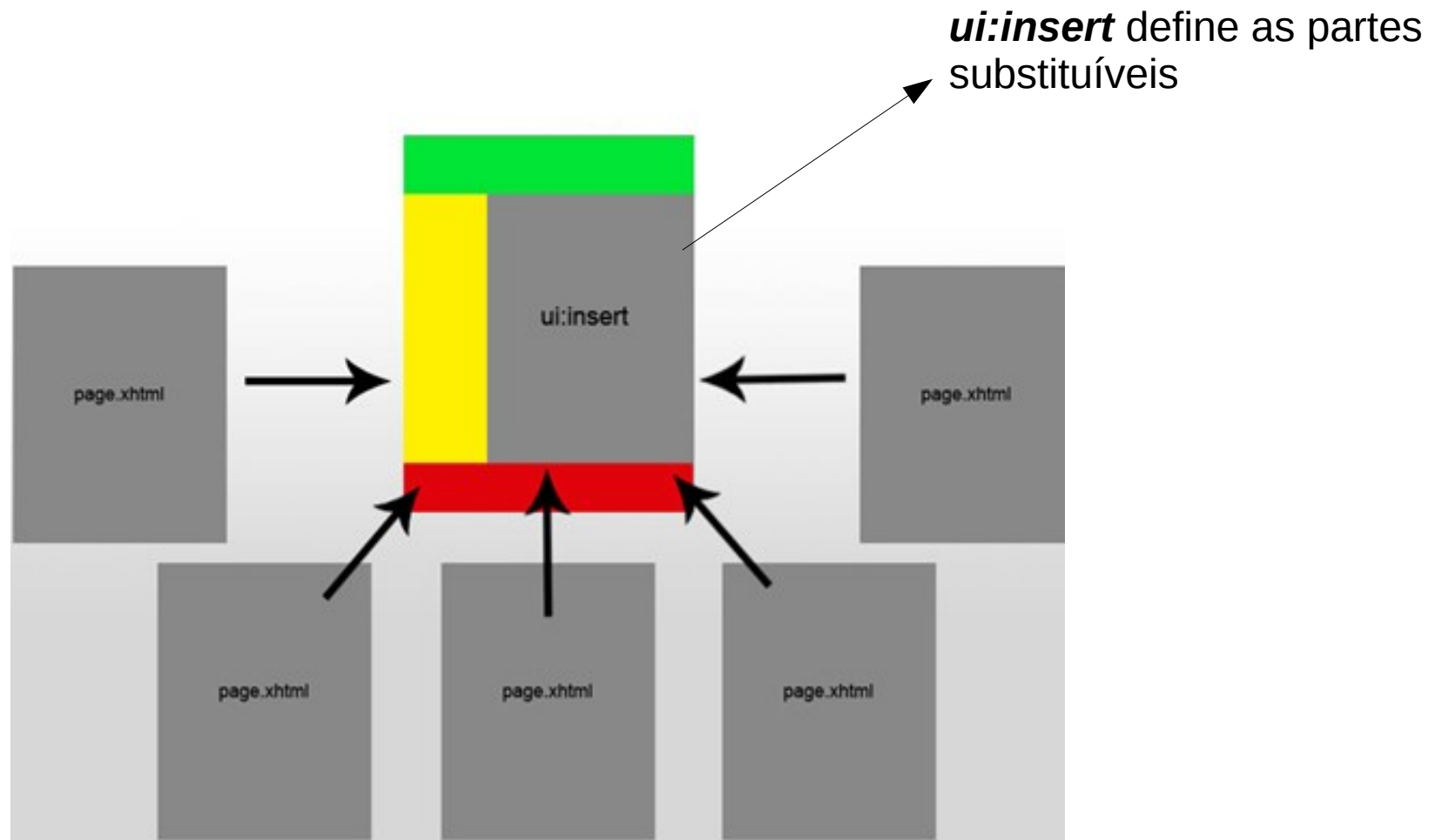
Recebe um `ActionEvent` como parâmetro

## XHTML

- Alta reutilização de código
- Até 50% mais rápido



# Templates



## Página de template criada layout.xhtml

***ui:include*** inclui o conteúdo de uma página.xhtml em outra página.xhtml

O componente incluso deve estar contido em um ***ui:composition***

A tag ***ui:insert*** serve como um container para definir que a área do template pode ser substituída

```
<h:body>
  <div id="pagina">

    <div id="header" class="border">
      <ui:insert name="header">
        <ui:include src="header.xhtml" />
      </ui:insert>
    </div>

    <div class="content">
      <div id="Left" class="Col_Left border">
        <ui:insert name="left">
          <ui:include src="left.xhtml" />
        </ui:insert>
      </div>

      <div id="content" class="Col-center border">
        <ui:insert name="content">
          <h2>Conteúdo a ser substituído</h2>
        </ui:insert>
      </div>
    </div>

    <div id="footer" class="border">
      <ui:insert name="footer">
        <ui:include src="footer.xhtml" />
      </ui:insert>
    </div>

  </div>
</h:body>
```

## Executando a página de template

**Topo da página**

**Lado esquerdo**

**Conteúdo a ser substituído**

**Rodapé da página**

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<h:html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head></h:head>

<h:body>
  <ui:composition template="layout.xhtml">
    <ui:define name="content">

      <ui:include src="content/content.xhtml" />

    </ui:define>
  </ui:composition>
</h:body>
</h:html>

```

## pagina1.xhtml

A tag **ui:define** define qual região do template será substituída.

Qualquer conteúdo incluso dentro da tag **ui:composition** será incluso quando outra página Facelets incluir a página contendo a tag.

## content.xhtml

Aqui fica o conteúdo da página. Podemos adicionar um link para uma segunda página que irá substituir a página1.

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<h:html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head></h:head>

<h:body>
  <ui:composition>
    <h1>Conteúdo inserido e substituível</h1>
    <p><a href="pagina2.xhtml">Next</a></p>
  </ui:composition>
</h:body>
</h:html>

```

## Executando a página1.xhtml

**Topo da página**

**Lado esquerdo**

**Conteúdo inserido e substituível**

[Next](#)

**Rodapé da página**

## *ui:debug*

UI Debug permite que você visualize no navegador informações sobre a árvore de componentes JSF e variáveis de escopo.

Para abrir a janela de debug, utilize o atalho padrão **CTRL + SHIT + D** ou ainda podemos selecionar outra tecla de atalho( **T** ao invés de **D**);

```
<h:body>
  <ui:debug hotkey="t"/>
  <h:dataTable id="table1" value="#{clienteData.clienteList}"
    var="clienteTable" border="1">
    <f:facet name="header">
      <h:outputText value="Lista de Clientes" />
    </f:facet>
    <h:column>
      <f:facet name="header">
        <h:outputText value="Cliente" />
      </f:facet>
      <h:outputText value="#{clienteTable.nome}" />
    </h:column>
    <h:column>
      <f:facet name="header">
        <h:outputText value="Código" />
      </f:facet>
      <h:outputText value="#{clienteTable.codigo}" />
    </h:column>
  </h:dataTable>
</h:body>
```

## Debug Output

/exemploDataTable.xhtml

+ Component Tree

+ Scoped Variables

18/11/2015 10:13:56 - Generated by Mojarra/Facelets



## *ui:param*

UI Param é usada para passar objetos como variáveis entre páginas Facelets

### Parâmetro “titulo”

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

  <ui:param name="titulo" value="JSF 2 - Facelets" />

</ui:composition>
```

Recebendo parâmetro no  
template.xhtml

```
<title>#{titulo}</title>
```

## Primefaces

### Adicionando as dependências no pom.xml

```
<repositories>
  <repository>
    <id>prime-repo</id>
    <name>PrimeFaces Maven Repository</name>
    <url>http://repository.primefaces.org</url>
    <layout>default</layout>
  </repository>
</repositories>
```

```
<dependency>
  <groupId>org.primefaces</groupId>
  <artifactId>primefaces</artifactId>
  <version>4.0</version>
</dependency>
```

```
<dependency>
  <groupId>org.primefaces.themes</groupId>
  <artifactId>all-themes</artifactId>
  <version>1.0.10</version>
</dependency>
```

## Primefaces

Configurando o tema no  
web.xml

```
<context-param>  
  <param-name>primefaces.THEME</param-name>  
  <param-value>bootstrap</param-value>  
</context-param>
```

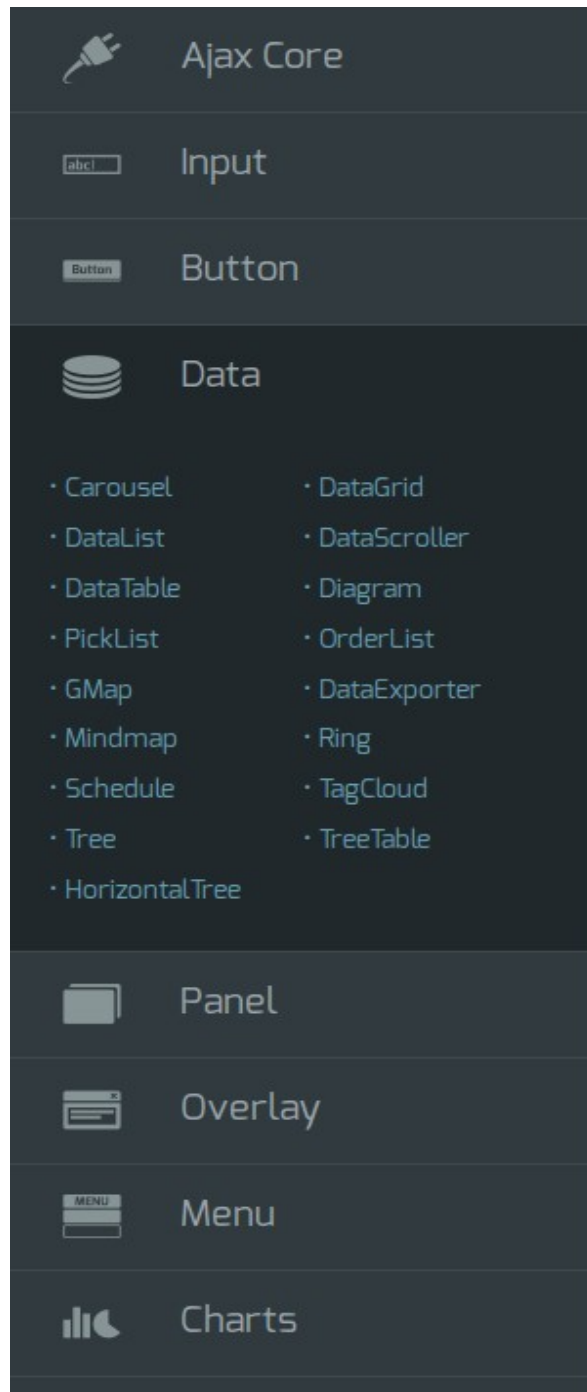


Tema escolhido

## Tag Library do primefaces

```
xmlns:p="http://primefaces.org/ui"
```

# Primefaces



<http://www.primefaces.org/showcase/>

**Primefaces** oferece uma documentação de fácil acesso e com exemplos de como usar cada um de seus componentes

## Alguns elementos do Primefaces

### Tabela de Dados

```
<p:dataTable id="primeTable" value="#{clienteData.clienteList}"
  var="cliente" style="width: 300px">
  <p:column headerText="Cliente" style="text-align: center">
    <h:outputText value="#{cliente.nome}" />
  </p:column>
  <p:column headerText="Código" style="text-align: center">
    <h:outputText value="#{cliente.codigo}" />
  </p:column>
</p:dataTable>
```

Cliente	Código
Cliente 01	1
Cliente 02	2
Cliente 03	3

## Alguns elementos do Primefaces

### Mensagens

```
@ManagedBean(name = "mensagens")
public class MensagensPrime {

    public void info() {
        FacesContext.getCurrentInstance()
            .addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO, "Info", "Mensagem informando algo!"));
    }

    public void erro() {
        FacesContext.getCurrentInstance()
            .addMessage(null, new FacesMessage(FacesMessage.SEVERITY_ERROR, "Erro!", "Ocorreu um erro!"));
    }


    public void aviso() {
        FacesContext.getCurrentInstance()
            .addMessage(null, new FacesMessage(FacesMessage.SEVERITY_WARN, "Aviso!", "Isso é um aviso!"));
    }

    public void fatal() {
        FacesContext.getCurrentInstance()
            .addMessage(null, new FacesMessage(FacesMessage.SEVERITY_FATAL, "Fatal!", "Parou de funcionar!"));
    }
}
```

```
<h:form>
    <p:messages id="messages" showDetail="true" autoUpdate="true" closable="true" />
    <p:commandButton value="Mensagem Informativa" actionListener="#{mensagens.info}" />
    <p:commandButton value="Mensagem de Erro" actionListener="#{mensagens.erro}" />
    <p:commandButton value="Mensagem de Aviso" actionListener="#{mensagens.aviso}" />
    <p:commandButton value="Mensagem Fatal" actionListener="#{mensagens.fatal}" />
</h:form>
```

## Alguns elementos do Primefaces

### Mensagens

 **Info** Mensagem informando algo!

Mensagem Informativa

Mensagem de Erro

Mensagem de Aviso

Mensagem Fatal

 **Aviso!** Isso é um aviso!

Mensagem Informativa

Mensagem de Erro

Mensagem de Aviso

Mensagem Fatal

 **Erro!** Ocorreu um erro!

Mensagem Informativa

Mensagem de Erro

Mensagem de Aviso

Mensagem Fatal

 **Fatal!** Parou de funcionar!

Mensagem Informativa

Mensagem de Erro

Mensagem de Aviso

Mensagem Fatal

## Alguns elementos do Primefaces

### Mensagens

```
<h:panelGrid id="panel" columns="3" cellpadding="5" style="">
  <p:outputLabel value="nome" for="nome"/>
  <p:inputText id="nome" required="true"/>
  <p:message for="nome" />
```

Mensagem padrão,  
com ícone e texto.

```
  <p:outputLabel value="sobrenome" for="sobrenome"/>
  <p:inputText id="sobrenome" required="true"/>
  <p:message for="sobrenome" display="text" />
```

Mensagem somente com  
texto.

```
  <p:outputLabel value="idade" for="idade"/>
  <p:inputText id="idade" required="true"/>
  <p:message for="idade" display="icon" />
```

Mensagem somente com  
ícone

```
  <p:outputLabel value="telefone" for="telefone"/>
  <p:inputText id="telefone" required="true"/>
  <p:message for="telefone" display="tooltip" />
</h:panelGrid>
```

Mensagem na forma de  
tooltip.



## Alguns elementos do Primefaces

### Mensagens



nome: Erro de validação: o valor é necessário. nome: Erro de validação: o valor é necessário.  
sobrenome: Erro de validação: o valor é necessário. sobrenome: Erro de validação: o valor é necessário.  
idade: Erro de validação: o valor é necessário. idade: Erro de validação: o valor é necessário.  
telefone: Erro de validação: o valor é necessário. telefone: Erro de validação: o valor é necessário.

nome \*



nome: Erro de validação: o valor é necessário.

sobrenome \*

sobrenome: Erro de validação: o valor é necessário.

idade \*




telefone \*

Enviar

## Alguns elementos do Primefaces

### Mensagens

```
<p:inputText id="nome" value="#{clientes.nome}">  
  <p:ajax event="keyup" update="output"/>  
</p:inputText>  
<h:outputText id="output" value="#{clientes.nome}" />
```



Teste Ajax!	<input type="text"/>	Teste Ajax!
-------------	----------------------	-------------

**<p:ajax>** para utilizarmos ajax no Primefaces.

**event:** indica qual evento no lado do cliente deve ativar as requisições.

**update:** indica o que será atualizado.

Há também o atributo **listener** que realiza alguma ação assim que a requisição é feita.