MyFaces Tomahawk

Rafael De Pauli Baptista (rafaelpbaptista@gmail.com)

Como podemos perceber nesses últimos anos, o desenvolvimento de aplicações web está em constante evolução. No mundo java, o primeiro conceito de páginas dinâmicas veio com os servlets, onde um classe java utiliza outputstream para escrever tags HTML para o browser interpretar. Logo em seguida surgiu as páginas JSP (que no fundo são servlets) para facilitar a vida do programador, não necessitando dessa forma escrever tags HTML utilizando outputstream. Com o surgimento do JSP, não demorou muito para a criação de um padrão (Design Pattern) que utiliza-se de forma inteligente o JSP e o servlet, surgindo assim o padrão MVC (Model View Controller) e Frameworks para tal padrão.

Um dos Frameworks MVC mais conhecidos atualmente é o Struts (http://struts.apache.org/), onde existe o conceito que uma classe (ActionForm) representa os dados de uma página JSP e outra classe (Action) que trata as requisições oriundas da página JSP.

Alguns anos atrás (2004) surgiu um novo paradigma de programação web, muito parecido com a programação de aplicações visuais (AWT e Swing). Tal paradigma é encontrado no Framework JSF (JavaServer Faces) . O conceito básico desse Framework é que existem classes que representam os componentes da sua página JSP e outras que tratam os eventos lançados por tais componentes (Classes Listeners).

O JSF é uma especificação (JSR 127 define o JSF 1.0 e o 1.1, JSR 252 define o JSF 1.2), com isto existindo algumas implementações. A implementação mais conhecida do JSF é o da SUN (http://java.sun.com/javaee/javaserverfaces/), onde engloba todo o núcleo do JSF. Existem componentes extras, que não são encontrados no núcleo do JSF, implementados por vários projetos e empresas. Esse artigo tratará dos componentes extras implementados pelo projeto Apache MyFaces (http://myfaces.apache.org/). Os componentes tratados aqui são do MyFaces Tomahawk.

Para a construção dos exemplos implementados nesse artigo, utilizaremos o IDE Eclipse 3.1.2 (http://download.eclipse.org/eclipse/downloads/drops/R-3.1.2-200601181600/index.php) juntamente com a versão free do plugin Exadel 3.6 (http://www.exadel.com/web/portal/download/es). Tal plugin irá nos auxiliar na criação de projetos JSF.

Como criar um projeto JSF

Para a criação de um projeto JSF utilizando o plugin Exadel, deve-se seguir os seguintes passos:

1 – No menu bar, acessar File->New->Project

2 – Aparecerá uma tela contendo os tipos de projetos que poderá ser criado. Selecione Exadel Studio->JSF-> JSF Project. Clique em next.

3 – Aparecerá uma tela onde deverá ser informado o nome do projeto (Project Name) e qual implementação do JSF o projeto irá utilizar (JSF Environment). No exemplo desse artigo o nome do projeto será **myFacesSample** e a implementação JSF será o **MyFaces 1.1.3**. Clique em next.

4 – A próxima tela deverá ser selecionado um servidor de aplicação web. Caso o combo Runtime não possuir nenhum servidor, clique no botão New, localizado ao lado do combo, para selecionar um servidor web instalado na máquina. Após selecionar um servidor do combo Runtime, selecione tal servidor no campo de Target Server. Caso não exista nenhuma opção no campo de Target Server, clique no botão New para criar um novo Target Server. Logo após a esses passos, clique em finish. Com isso um projeto JSF será criado.

Como criar uma página JSP

Antes de criar uma página JSP, abra a perspectiva do Exadel no seu Eclipse. Para isso, no menu bar acesse Window->Open Perspective->Other->Exadel Studio. Com isso a perspectiva do Exadel estará aberta.

Crie uma nova página JSP sob a pasta WebContent do seu projeto JSP. Esse é o diretório raiz de sua aplicação web. Para criar um nova página JSP, clique com o botão direito do mouse sob a pasta WebContent e acesse New-> JSP File. Com isso aparecerá uma janela de dialogo para a criação da página JSP. Informe o nome da página e o Template. Dica: Selecione o Template JSFBasePage, que com isso será criado um JSP básico com as tags mínimas que uma página deverá conter.

Importante: O Template JSFBasePage não adiciona a tag <h:form>. A maioria das páginas necessitam dessa tag para a criação de um form para a sua página.

Ao abrir uma página JSP, do lado direito do editor, aparecerá uma paleta de componentes que podem ser adicionados ao seu JSP.

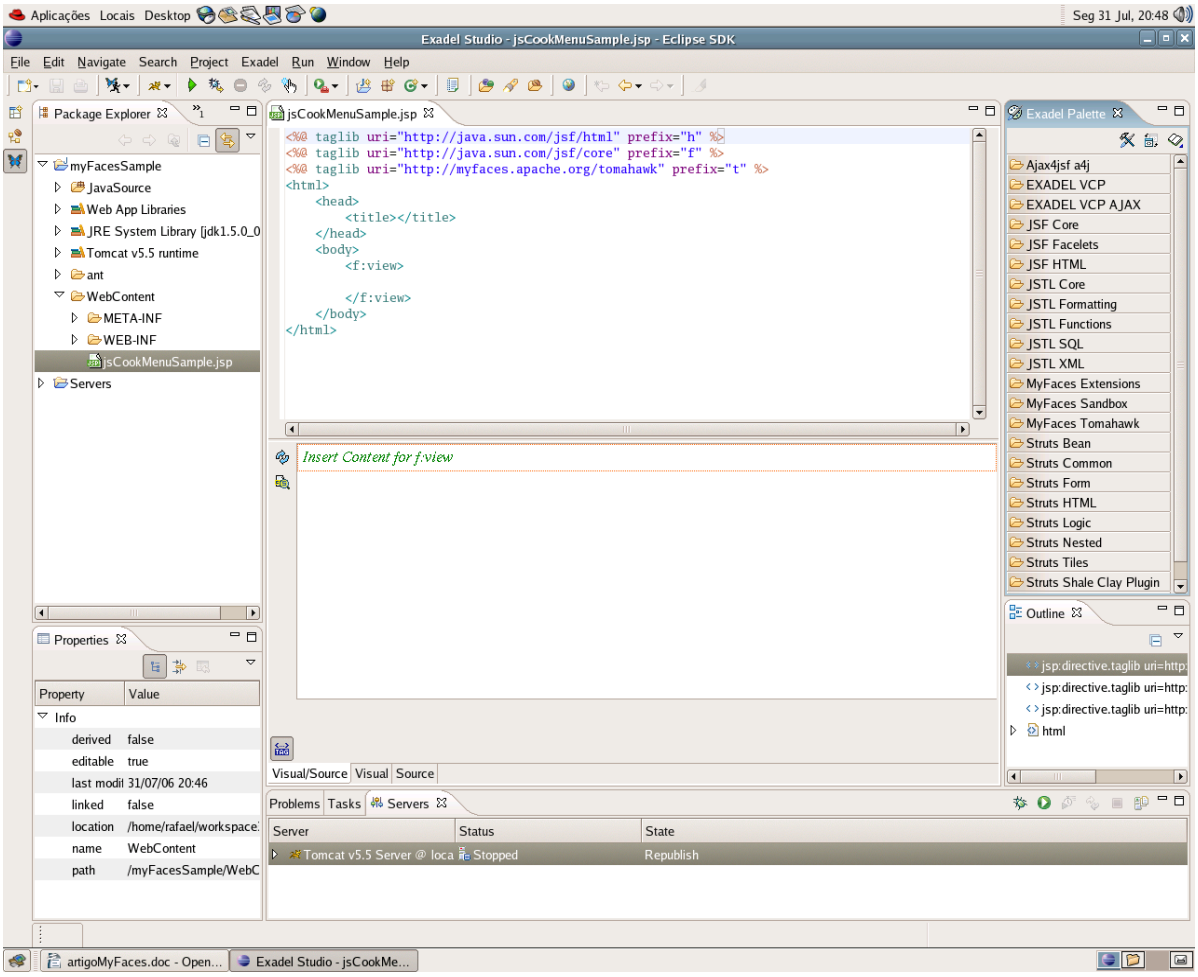

Figura 1: Paleta de componentes JSF.

Componentes MyFaces Tomahawk

JSCookMenu <t:jscookMenu>

A tag jscookMenu redenriza um menu javascript. Os itens que constitui o

menu são redenrizados através das tags NavigationMenuItem e NavigationMenuItems.

Os itens do menu podem ser estáticos ou dinâmicos.

Serão apresentados exemplos com os itens de menu estáticos e dinâmicos.

Segue sintaxe desse componente:

```
<t:jscookMenu [ user-role-support-attributes ]

     [layout="values {hbr, hbl, hur, hul, vbr, vbl, vur, vul}"]
     [theme="values {ThemeIE, ThemeMiniBlack, ThemeOffice, ThemePanel}"]

     Nested <t:navigationMenuItem> or <t:navigationMenuItems> tags (menu items)
</t:jscookMenu>
```

Menu com itens estáticos

Seleciona na paleta de componentes MyFaces->Tomahawk->jscookMenu. Com isso aparecerá um caixa de dialogo para a criação de um jsCookMenu. Os campos para a inserção de dados são:

- Layout: Podendo ser qualquer um dos valores descritos na sintaxe do componente.
- Theme: Podendo ser qualquer um dos valores descritos na sintaxe do componente.

No nosso exemplo, foi criado e seguinte menu:

```
<t:jscookMenu layout="hbr" theme="ThemeOffice">

</t:jscookMenu>
```

OBS: Os layouts começados com a letra “H” são layouts horizontais e os começos com a letra “V” são layouts verticais.

Para acrescentar itens de menu estáticos ao seu jsCookMenu, utilize a tag t:navigationMenuItem.

Segue um exemplo de um menu estático:

```
<t:jscookMenu layout="hbr" theme="ThemeOffice" >
    <t:navigationMenuItem itemLabel="PRIMEIRO MENU" icon="/imagens/myfaces.gif">
        <t:navigationMenuItem itemLabel="PRIMEIRO NIVEL" action="go_primeiroNivel" icon="/imagens/myfaces.gif"/>
        <t:navigationMenuItem itemLabel="SEGUNDO NIVEL" action="go_segundoNivel" icon="/imagens/myfaces.gif"/>

        <t:navigationMenuItem itemLabel="TERCEIRO NIVEL COM DIVISORIA" split="true" icon="/imagens/myfaces.gif">
                <t:navigationMenuItem itemLabel="OPCAO TERCEIRO NIVEL" action="ALGUMA_ACAO"
icon="/imagens/myfaces.gif"/>
        </t:navigationMenuItem>

   </t:navigationMenuItem>

   <t:navigationMenuItem itemLabel="SEGUNDO MENU" action="ALGUMA_ACAO" icon="/imagens/myfaces.gif"/>

</t:jscookMenu>
```

Descrição da tag navigationMenuItem:

- itemLabel: Texto que aparecerá na tela. Label do item.
- Action: Ação que será chamada quando o item receber um clique do mouse. Tal ação deve estar declarada no seu arquivo faces-config.xml.
- Icon: Imagem que aparecerá ao lado da label do item. O caminho da imagem deve começar pela raiz da aplicação, que no nosso exemplo é o diretório WebContent do nosso projeto.

O arquivo faces-config.xml descreve todas as navegações entre as páginas e os Managed Bens utilizados pelo sistema. Abaixo segue um exemplo do arquivo faces-config.xml para a execução do menu.

```
<?xml version="1.0"?>
<!DOCTYPE faces-config PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
                              "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
<faces-config>
        <navigation-rule>
                <!-- Ações utilizadas pelo exemplo jscookMenuSample. As ações possuem as URL's que podem ser
chamadas pelo Menu -->
                <navigation-case>
```

```
                <from-outcome>go_primeiroNivel</from-outcome>
                <to-view-id>/primeiroNivel.jsp</to-view-id>
            </navigation-case>
            <navigation-case>
                <from-outcome>go_segundoNivel</from-outcome>
                <to-view-id>/segundoNivel.jsp</to-view-id>
            </navigation-case>
        </navigation-rule>
</faces-config>
```

Um t:navigationMenuItem pode possuir tags filhas, onde tais tags são sub-itens da tag pai. No nosso exemplo, a tag com o itemLabel igual a “TERCEIRO NIVEL COM DIVISORIA” possui um sub-item.

A figura abaixo mostra o jsCookMenuSample rodando.

ATENÇÃO: Para acessar a página jsCookMenuSample deve ser utilizado a seguinte URL: http://127.0.0.1:8080/myFacesSample/jsCookMenuSample.jsf.

PRESTE ATENÇÃO, NÃO É jsCookMenuSample.jsp e sim jsCookMenuSample.jsf.

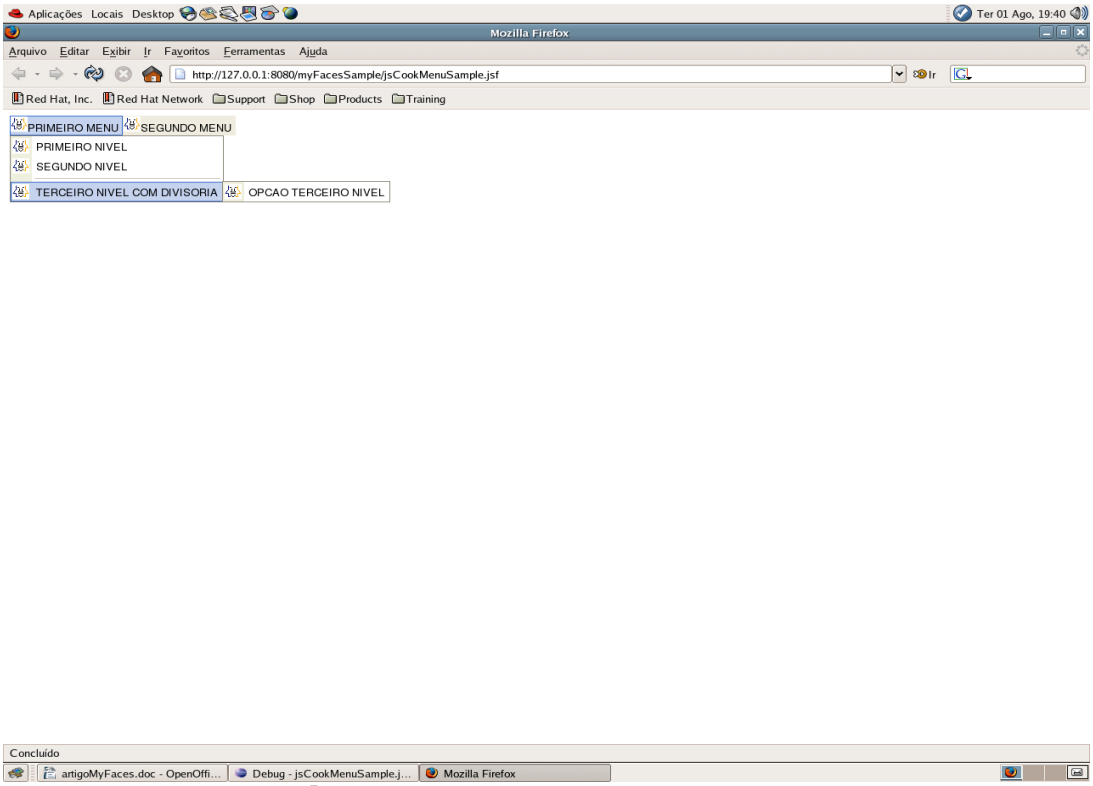

Figura 2 – Exemplo JSCookMenu

Menu com itens dinâmicos

A criação de um menu dinâmico é feita através de uma classe, onde tal classe possuí, como propriedade, um array de NavigationMenuItem. Tal array de NavigationMenuItem é renderizado na página através da tag t:navigationMenuItems.

Segue exemplo de como montar um jsCookMenu dinâmico.

A sua página JSP deverá conter um trecho parecido com:

```
<t:jscookMenu layout="hbr" theme="ThemeOffice">
        <t:navigationMenuItems value="#{dynamicMenuTest.navItems}" />
</t:jscookMenu>
```

A classe dynamicMenuTest tem que estar definida no arquivo faces-config.xml e a propriedade navItems deve ser um array de NavigationMenuItem (NavigationMenuItem []).

Segue código fonte da classe DynamicMenuTest, onde o menu criado no exemplo de menu com itens estáticos é criado agora dentro da classe DynamicMenuTest.

```
package sample;


import java.util.ArrayList;


import org.apache.myfaces.custom.navmenu.NavigationMenuItem;


/**
 * @author Rafael De Pauli Baptista
 */
public class DynamicMenuTest {
        private NavigationMenuItem[] navItems;


        public DynamicMenuTest() {
                // Criando os Menus Raizes
                NavigationMenuItem primeiroMenu = new NavigationMenuItem("Primeiro
Menu",null,"/imagens/myfaces.gif",false);
                NavigationMenuItem segundoMenu = new NavigationMenuItem("Segundo
Menu","ALGUMA_ACAO","/imagens/myfaces.gif",false);


                // Agora acrescentando os menus raízes ao array que será retornado ao JSP
                this.navItems = new NavigationMenuItem[2];
                this.navItems[0] = primeiroMenu;
                this.navItems[1] = segundoMenu;


                // Agora acrescentando os itens no menu primeiroMenu
                ArrayList subItensPrimeiroMenu = new ArrayList();
                subItensPrimeiroMenu.add(new NavigationMenuItem("PRIMEIRO
NIVEL","go_primeiroNivel","/imagens/myfaces.gif",false));
                subItensPrimeiroMenu.add(new NavigationMenuItem("SEGUNDO
NIVEL","go_segundoNivel","/imagens/myfaces.gif",false));
                NavigationMenuItem terceiroSubItem = new NavigationMenuItem("TERCEIRO
NIVEL",null,"/imagens/myfaces.gif",true);
                subItensPrimeiroMenu.add(terceiroSubItem);
                primeiroMenu.setNavigationMenuItems(subItensPrimeiroMenu);


                // Agora acrescentando os sub-itens no item TERCEIRO NIVEL
                ArrayList subItensTerceiroNivel = new ArrayList();
                subItensTerceiroNivel.add(new NavigationMenuItem("OPCAO TERCEIRO
NIVEL","ALGUMA_ACAO","/imagens/myfaces.gif",false));
                terceiroSubItem.setNavigationMenuItems(subItensTerceiroNivel);
        }


        public NavigationMenuItem[] getNavItems() {
                return navItems;
        }


        public void setNavItems(NavigationMenuItem[] navItems) {
                this.navItems = navItems;
        }
}
```

Tree2 <t:tree2>

O componente Tree2 redenriza uma árvore de dados utilizando uma table HTML. Tal árvore é dinâmica, expandindo e encapsulando os dados através de cliques do usuário.

O componente suporta interação tanto do lado do cliente como do lado do servidor. Isso pode ser configurado através da propriedade clientSideToggle. O valor padrão de tal propriedade é true, sendo assim a interação é realizada do lado do cliente, através de javascript. Caso queira realizar a interação através do servidor, coloque o valor da propriedade clientSideToggle=”false”. A interação do lado do servidor pode ser vantajosa quando a árvore possuir um volume muito grande de dados (nodos).

Os dados da árvore são obtidos apartir de um método de alguma classe. No nosso exemplo, o método getTreeNodes da classe Tree2Sample retorna um objeto do tipo TreeNode, que será utilizado pela página para a criação da árvore.

Abaixo segue trecho da página JSP que é responsável pela renderização da árvore.

```
<t:tree2 value="#{tree2Sample.treeNodes}" var="node">
     <f:facet name="nodo">
        <h:panelGroup>
           <h:outputText value="#{node.description}"/>
        </h:panelGroup>
     </f:facet>

     <f:facet name="documento">
          <h:panelGroup>
                <h:commandLink id="link" action="#{tree2Sample.submit}" actionListener="#{tree2Sample.process}"
immediate="true">
                      <h:outputText value="#{node.description}"/>
                </h:commandLink>
          </h:panelGroup>
     </f:facet>
</t:tree2>
```

Logo abaixo será mostrado o código fonte da classe Tree2Sample responsável pela criação do objeto TreeNode utilizado para a renderização da árvore. Tal classe deve estar definida no arquivo faces-config.xml.

```
package sample;


import org.apache.myfaces.custom.tree2.TreeNode;
import org.apache.myfaces.custom.tree2.TreeNodeBase;


import java.util.Iterator;


import javax.faces.component.UIComponent;
import javax.faces.component.html.HtmlOutputText;
import javax.faces.event.ActionEvent;



public class Tree2Sample {
    private String acaoSubmit;


    public TreeNode getTreeNodes() {

        // Definicao do construtor: TreeNodeBase (java.lang.String type, java.lang.String description, boolean
leaf)
        TreeNodeBase primeiroNodo = new TreeNodeBase("nodo", "Primeiro nodo", false);

         primeiroNodo.getChildren().add(new TreeNodeBase("documento", "Primeira Opcao Primeiro nodo",
"go_primeiroNivel", true));


        TreeNodeBase segundoNodo = new TreeNodeBase("nodo", "Segundo nodo", false);
        segundoNodo.getChildren().add(new TreeNodeBase("documento", "Primeira Opcao Segundo nodo",
"go_segundoNivel", true));


        TreeNodeBase nodoRaiz = new TreeNodeBase("nodo", "Raiz", false);
```

```
        nodoRaiz.getChildren().add(primeiroNodo);
        nodoRaiz.getChildren().add(segundoNodo);


        return nodoRaiz;
    }


    public String submit() {
        return acaoSubmit;
    }


    public void process(ActionEvent event) {
        UIComponent component = (UIComponent) event.getSource();


        Iterator iterator = component.getFacetsAndChildren();


        while (iterator.hasNext()) {
            Object object = iterator.next();


            if (object instanceof HtmlOutputText) {
                HtmlOutputText htmlOutputText = (HtmlOutputText) object;


                // Comparando o valor com a descricao dos nodos
                if (htmlOutputText.getValue().equals("Primeira Opcao Primeiro nodo")) {
                    this.acaoSubmit = "go_primeiroNivel";
                } else if (htmlOutputText.getValue().equals("Primeira Opcao Segundo nodo")) {
                    this.acaoSubmit = "go_segundoNivel";
                } else {
                    this.acaoSubmit = "";
                }
            }
        }
    }
}
```

Importante salientar alguns tópicos:

1 – Quando o link (<h:commandLink>) recebe um clique do usuário, o método process da classe Tree2Sample é invocado antes da página fazer o submit. O método process retorna uma string que determina qual ação deve ser chamada pela página JSP. Tal ação é determinada pelo arquivo faces-config.xml através da tag <navigation-case>.

2 – Uns dos construtores da classe TreeNodeBase possui os seguintes parâmetros: java.lang.String type, java.lang.String description, boolean leaf

3 - DEVE EXISTIR NO SEU JSP UM FACET COM O MESMO NOME DO ID DA CLASSE TreeNodeBase INSTANCIADA NA SUA CLASSE. Caso isso não ocorra, um erro será gerado.

A figura abaixo mostra o resultado do exemplo tree2Sample.

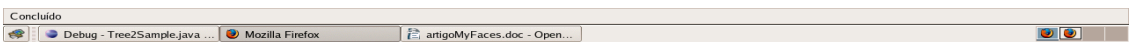

Figura 3 – Exemplo Tree2

File Update <t:inputFileUpdate/>

O componente <t:inputFileUpdate> redenriza um HTML input type=”file” na página JSP, para que o cliente possa fazer uploads de arquivos no servidor. Entende-se por upload a operação de submeter arquivos para o servidor web.

A classe utilizada para manipular as informações de tal componente é a classe org.apache.myfaces.custom.fileupload.UploadedFile, onde tal classe fornece vários métodos para a manipulação do arquivo enviado. O método mais importante dessa classe é o método getInputStream, onde podemos obter um inputStream e com isso trabalhar com o array de bytes que compõem o arquivo. Com o array de bytes em mãos, podemos gravar o arquivo em disco ou manipula-lo em memória.

O arquivo web.xml da aplicação criada através do plugin Exadel, possui alguns parâmetros importantes utilizado pelo componente <t:inputFileUpdate>, sendo elas:

- uploadMaxFileSize: Tamanho máximo permitido do arquivo a ser enviado pelo cliente.
- UploadThresholdSize: Tamanho máximo do arquivo que permanece em memória após o upload. Arquivos que ultrapassarem esse limite serão gravados em disco.

Abaixo segue a sintaxe do componente:

```
<t:inputFileUpload
      accept = “VALOR”
      Storage = ”VALOR”
      Value = ”VALOR”/>
```

Onde:

- accept: Valor que indica um filtro para o arquivo.
- Storage: Indica o método de “salvamento” do arquivo, podendo conter os valores Memory (padrão) e File.
- Value: Propriedade de uma classe que é do tipo org.apache.myfaces.custom.fileupload.UploadedFile.

Abaixo segue um exemplo de uma página JSP que utiliza o componente <t:inputFileUpdate>.

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html>
<head>
<title></title>
</head>
<body>
<f:view>
        <h:form enctype="multipart/form-data">
                Arquivo para upload no servidor:
                <t:inputFileUpload required="true"
                                                        storage="file"
                                                        value="#{fileUploadSample.uploadedFile}"
                                                        accept="/tmp/*"/>
```

```
                <h:commandButton value="Upload" action="#{fileUploadSample.processUpload}"/>
        </h:form>
</f:view>
</body>
</html>
```

Agora segue o código fonte da classe responsável pela manipulação do arquivo. Tal classe deve estar definida no arquivo faces-config.xml.

O arquivo enviado pelo cliente é salvo no servidor.

```
package sample;


import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;


import org.apache.myfaces.custom.fileupload.UploadedFile;


public class FileUploadSample {
    // Classe que representa o arquivo enviado pelo cliente (arquivo do upload)
    private UploadedFile uploadedFile;


    public FileUploadSample() {
    }


    /**
     * Obtem a classe que manipula o arquivo enviado pelo cliente
     * @return
     */
    public UploadedFile getUploadedFile() {
        return uploadedFile;
    }


    /**
     * Seta a classe que manipula o arquivo enviado pelo cliente
     * @param uploadedFile
     */
    public void setUploadedFile(UploadedFile uploadedFile) {
        this.uploadedFile = uploadedFile;
    }


        /**
         * Faz processamento do arquivo enviado pelo cliente
         * @return
         */
        public String processUpload() {
        try {

                //Criando arquivo no servidor
```

```
                File file = new File(uploadedFile.getName());


                BufferedInputStream bufferedInputStream = new
                                        BufferedInputStream(this.uploadedFile.getInputStream());


                FileOutputStream fileOutputStream = new FileOutputStream(file);


                // Salvando o arquivo
                try {
                        byte[] buffer = new byte[1024];
                        int count;
                        while ((count = bufferedInputStream.read(buffer)) > 0)
                                fileOutputStream.write(buffer, 0, count);
                } finally {
                        bufferedInputStream.close();
                        fileOutputStream.close();
                }
        } catch (IOException exception) {
                exception.printStackTrace();
        }


        return null;
    }
}
```

<t:panelTabbedPane/>

O componente <t:panelTabbedPane/> redenriza um painel com tabs. Cada tab é criada através da tag <t:panelTab>. Abaixo segue o uso do componente.

```
<t:panelTabbedPane selectedIndex="int"
                    activeTabStyleClass="CSSClass"
                    inactiveTabStyleClass="CSSClass"
                    disabledTabStyleClass="CSSClass"
                    activeSubStyleClass="CSSClass"
                    inactiveSubStyleClass="CSSClass"
                    tabContentStyleClass="CSSClass">
    <t:panelTab ...>
        ...(anyComponents)...
    </t:panelTab>
</t:panelTabbedPane>
```

Segue sintaxe do componente:

```
<t:panelTabbedPane
        selectedIndex = “VALOR”
        activeTabStyleClass = “VALOR”
        inactiveTabStyleClass = “VALOR”
        disabledTabStyleClass = “VALOR”
        activeSubStyleClass = “VALOR”
        inactiveSubStyleClass = “VALOR”
        tabContentStyleClass = “VALOR”/>
```

Onde:

- selectedIndex: Índice da tab que é selecionada por padrão.
- ActiveTabStyleClass: Style Class (css) aplicado na célula de uma tab ativa.
- InactiveTabStyleClass: Style Class (css) aplicado na célula de uma tab inativa.
- DisabledTabStyleClass: Style Class (css) aplicado na célula de uma tab desabilitada.
- ActiveSubStyleClass: Style Class (css) aplicado na célula de uma sub-tab ativa.
- InactiveSubStyleClass: Style Class (css) aplicado na célula de uma sub-tab inativa.
- TabContentStyleClass: Style Class (css) aplicado na célula de uma tab contendo células.

Obs: O primeiro índice da tab do panelTabbedPane é igual a zero.

O componente é relativamente de fácil uso. Cada tab (<t:panelTab>) pode possuir vários componentes, como por exemplo vários inputTexts.

Caso algum erro ocorra no uso do componente, isso geralmente é conseqüência de falta de alguma configuração no arquivo web.xml.

Abaixo segue um exemplo de uma página JSP contendo o componente <t:panelTabbedPane/>

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html>
<head>
<title></title>
</head>
<body>
<f:view>
        <h:form>
                <t:panelTabbedPane serverSideTabSwitch="false">
                        <t:panelTab label="Tab No.1">
                                <h:inputText value="#{tabbedPaneSample.valorInputTab1}" />
                        </t:panelTab>
                        <t:panelTab label="Tab No.2">
                                <h:inputText value="#{tabbedPaneSample.valorInputTab2}" />
                        </t:panelTab>
                </t:panelTabbedPane>

                <br>
                <h:commandButton value="Submeter"
                        action="#{tabbedPaneSample.processAction}" />

                <br>
                <h:outputText value="#{tabbedPaneSample.valorPrimeiroLabel}" />
                <br>
                <h:outputText value="#{tabbedPaneSample.valorSegundoLabel}" />
        </h:form>
</f:view>
</body>
</html>
```

Abaixo segue o código fonte do Managed Bean responsável pela página JSP descrita acima. Tal classe deve estar definida no arquivo faces-config.xml.

```
package sample;

public class TabbedPaneSample {

    private String valorInputTab1;

    private String valorInputTab2;


    public TabbedPaneSample() {
```

```
}

/**
 * Obtem valor do input text da primeira aba do tabbed pane
 * @return
 */
public String getValorInputTab1() {
    return valorInputTab1;
}

/**
 * Seta valor do input text da primeira aba do tabbed pane
 * @param valorInputTab1
 */
public void setValorInputTab1(String valorInputTab1) {
    this.valorInputTab1 = valorInputTab1;
}

/**
 * Obtem valor do input text da segunda aba do tabbed pane
 * @return
 */
public String getValorInputTab2() {
    return valorInputTab2;
}

/**
 * Seta valor do input text da segunda aba do tabbed pane
 * @param valorInputTab2
 */
public void setValorInputTab2(String valorInputTab2) {
    this.valorInputTab2 = valorInputTab2;
}

/**
 * Obtem o valor do label que aparece logo abaixo do tabbed pane
 * @return
 */
public String getValorPrimeiroLabel() {
    if (this.valorInputTab1 != null && this.valorInputTab1.trim().length() > 0) {
            return "Valor Input Text Primeito Label: " + this.valorInputTab1;
    } else {
            return "";
    }
}

/**
 * Obtem o valor do label que aparece logo abaixo do tabbed pane
 * @return
 */
public String getValorSegundoLabel() {
    if (this.valorInputTab2 != null && this.valorInputTab2.trim().length() > 0) {
```

```
                return "Valor Input Text Segundo Label: " + this.valorInputTab2;
        } else {
                return "";
        }
    }

    public String processAction() {
        return null;
    }
}
```

Abaixo seque figura exibindo a tela sendo executada.

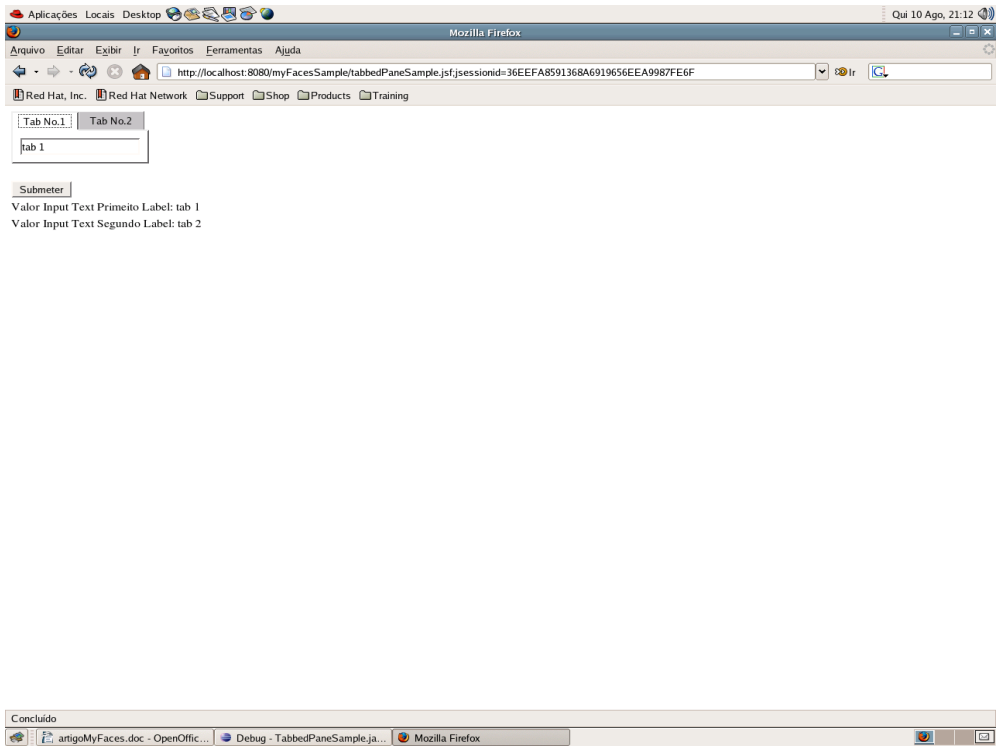

Figura 4 – Exemplo panelTabbedPane

<t:inputCalendar/>

O componente <t:inputCalendar/> renderiza na tela um componente javascript calendário aonde é possível escolher alguma data dentro do nosso calendário. Tal componente calendário permite realizar a navegação entre os dias, meses e anos do nosso calendário.

Abaixo segue sintaxe do componente:

```
<t:inputCalendar/>

        monthYearRowClass = “VALOR”
        weekRowClass = “VALOR”
        dayCellClass = “VALOR”
        currentDayCellClass = “VALOR”
        renderAsPopup = “VALOR”
        addResources = “VALOR”
        popupButtonString = “VALOR”
        popupButtonStyle = “VALOR”
        popupButtonStyleClass = “VALOR”
        popupGotoString = “VALOR”
        popupTodayString = “VALOR”
        popupWeekString = “VALOR”
        popupScrollLeftMessage = “VALOR”
        popupScrollRightMessage = “VALOR”
        popupSelectMonthMessage = “VALOR”
```

```
        popupSelectYearMessage = “VALOR”
        popupSelectDateMessage = “VALOR” />
```

Onde:

- monthYearRowClass: Style Class (css) aplicado no cabeçalho aonde é mostrado o ano e mês do calendário.
- WeekRowClass: Style Class (css) aplicado no cabeçalho aonde é mostrado os dias da semana do calendário.
- DayCellClass: Style Class (css) aplicado nas células aonde são mostrados os dias do mês.
- CurrentDayCellClass: Style Class (css) aplicado na célula da data selecionada.
- RenderAsPopup: Renderiza uma entrada de data através de um pop-up javascript.
- AddResources: Indica para carregar os script e Style Class (css) padrões para a criação do pop-up javascript de input de data. Caso queira fornecer seus próprios scripts e Style Class, informe falso para essa propriedade.
- PopupDateFormat: Define um formato de data para o pop-up javascript de input de data.
- PopupButtonString: Informa um label para o botão que abre o pop-up javascript para o input da data. O valor padrão é '...'.
- popupButtonStyle: Style Class (css) aplicado no botão que abre o pop-up javascript para input de data.
- PopupButtonStyleClass: Style Class (css) aplicado no botão que abre o pop-up javascript para input de data.
- PopupGotoString: Seta a string para a frase padrão “Go To Current Month”.
- popupTodayString: Seta a string para a frase padrão “Today is”.
- PopupWeekString: Seta a string para a frase padrão “WK”.
- PopupScrollLeftMessage: Seta uma string para o scrolling do lado esquerdo.
- PopupScrollRightMessage: Seta uma string para o scrolling do lado direito.
- PopupSelectMonthMessage: Seta a string para a frase padrão “Click to select a month”.
- PopupSelectYearMessage: Seta a string para a frase padrão “Click to select a year”.
- PopupSelectDateMessage: Seta a string para a frase padrão “Select [date] as date”. Nota: Não retire a string [date] da frase, para que seja mostrado a data corrente selecionada na frase.
- RenderPopupButtonAsImage: [true|false] Se true redenziza uma imagem de calendário ao invés do botão. O valor padrão é false.

Abaixo segue fonte de uma página JSP contendo o componente <t:inputCalendar/>

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

<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html>
<head>
```

```
<title></title>
</head>
<body>
<f:view>
        <h:form>
                <t:inputCalendar value="#{calendarSample.date}"/><br><br>

                <t:inputCalendar value="#{calendarSample.dateSecond}"
                                                renderAsPopup="true"
                                                popupDateFormat="dd/MM/yyyy"/><br><br>

                <h:commandButton value="Submeter" action="#{calendarSample.processAction}"/>
        </h:form>
</f:view>
</body>
</html>
```

Abaixo segue o código fonte do Managed Bean responsável pela página JSP descrita acima. Tal classe deve estar definida no arquivo faces-config.xml.

```
package sample;


import java.text.SimpleDateFormat;
import java.util.Date;
public class CalendarSample {
    private Date date;
    private Date dateSecond;
    private SimpleDateFormat formatDate = new SimpleDateFormat("dd/MM/yyyy");


    public CalendarSample() {
    }


    public Date getDateSecond() {
        return dateSecond;
    }


    public void setDateSecond(Date dateSecond) {
        this.dateSecond = dateSecond;
    }


    public Date getDate() {
        return date;
    }


    public void setDate(Date date) {
        if (date != null) {
            this.date = date;


            System.out.println("Data informada: " + formatDate.format(date));
        }
    }


    public String processAction() {
         if (this.dateSecond != null) {
                System.out.println("Segunda data informada: " + formatDate.format(dateSecond));
         }


        return null;
```

```
    }
}
```

Segue figura demonstrando a execução da página JSP.

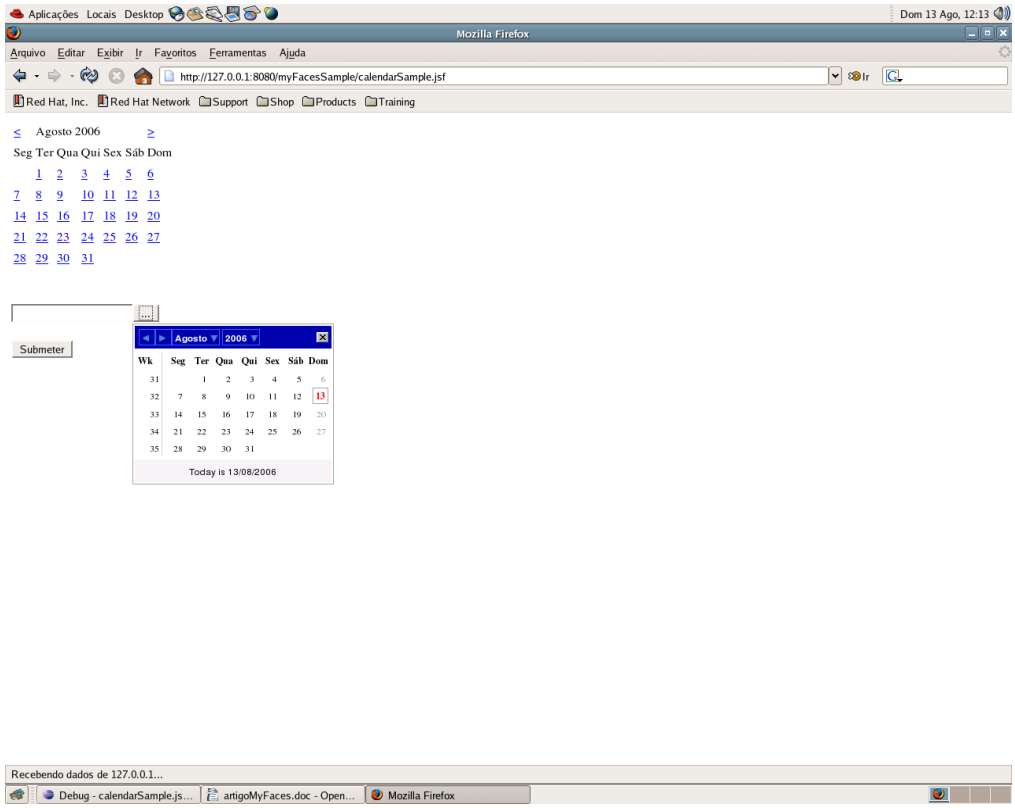

Figura 5 – Exemplo input Calendar

<t:popup>

Renderiza um popup no evento onmouseover sobre um texto.

Segue sintaxe do componente:

```
<t:popup

   styleClass = “VALOR”
   style = ”VALOR”

  displayAtDistanceX = “VALOR”
   displayAtDistanceX = “VALOR”
   closePopupOnExitingElement = “VALOR”
   closePopupOnExitingPopup = “VALOR” />
```

Onde:

- styleClass: styleClass para o pop-up. ATENÇÃO: Não sobrescreva a propriedade position: absolute e a propriedade display : none.
- Style: Estilo para o pop-up. ATENÇÃO: Não sobrescreva a propriedade position: absolute e a propriedade display : none.
- DisplayAtDistanceX: Distancia do eixo X (em pixel) que irá aparecer pop-up em relação ao lugar onde ocorreu o evento onmouseover.
- DisplayAtDistanceX: Distancia do eixo Y (em pixel) que irá aparecer pop-up em relação ao lugar onde ocorreu o evento onmouseover.

- ClosePopupOnExitingElement: [true|false]. Fecha o pop-up quando ou mouse deixa o elemento.
- ClosePopupOnExitingPopup: [true|false]. Fecha o pop-up quando o mouse deixa o pop-up.

Abaixo segue exemplo de uma página JSP contendo o componente <t:popup/>.

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t"%>
<html>
<head>
<title></title>
</head>
<body>
<f:view>
        <h:form>
                <t:popup>
                        <h:outputText value="Teste de Pop-up" />
                        <f:facet name="popup">
                                <h:panelGroup>
                                        <h:panelGrid columns="1">
                                                <h:outputText value="Valor do pop-up!!!!" />
                                        </h:panelGrid>
                                </h:panelGroup>
                        </f:facet>
                </t:popup>
        </h:form>
</f:view>
</body>
</html>
```

Segue figura mostrando o resultado do JSP.

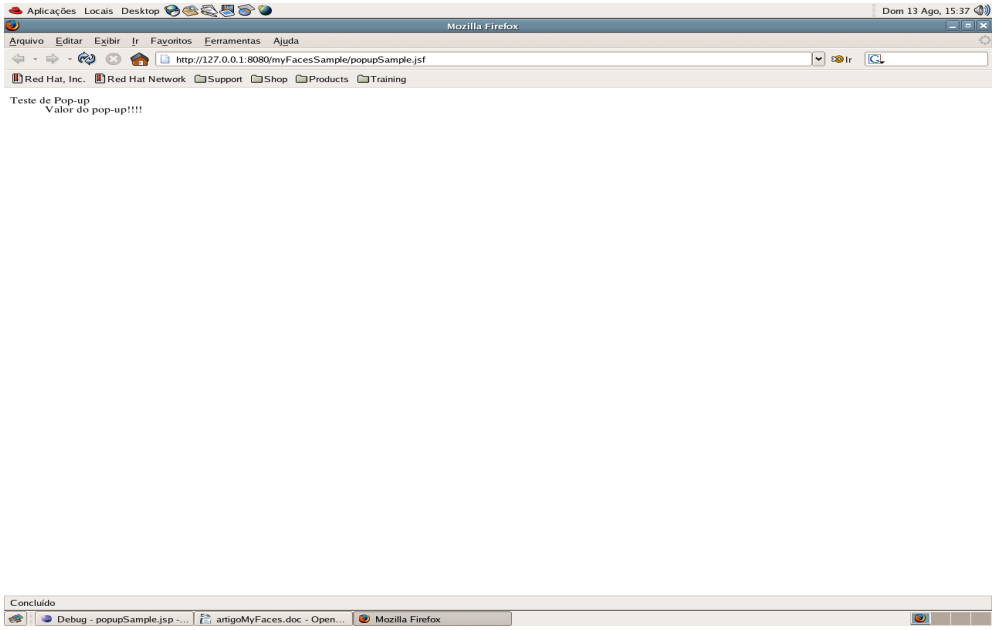

Figura 6 – Exemplo pop-up

<t:jsValueChangeListener>

Esse componente executa a função javascript onchange de algum componente JSF. Podemos utiliza-lo para, por exemplo, quando o valor de um componente A mudar, podemos executar alguma ação em algum outro componente.

Abaixo segue sintaxe do componente:

```
            <t:jsValueChangeListener

                for – O id do compoente alvo, quando o evento onchage se realizado
                expressionValue – O javascript que será executado. A palavra '$srcElem' é o alias utilizado para
acessar o componente que gerou a ação onchange e a palavra '$destElem' é o alias para o componente alvo, que está
especificado na propriedade 'for'
                property(opcional) – O resultado da tag expression é colocado na propriedade do componente alvo,
indicado na propriedade 'for'
                bodyEventTag(optional) – O evento é acionado quando ocorre o onchange. Mais eventos podem ser
colocados para acionar o componente, como por exemplo o onload. />
```

Exemplos:

Exemplo 1: Nesse exemplo, quando o valor do componente input text 1 é alterado, o valor dele é copiado para o componente input text 2.

```
<h:inputText id="text1">
   <t:jsValueChangeListener for="text2" property="value" expressionValue="$srcElem.value" />
</h:inputText>
<h:inputText id="text2"/>
```

Exemplo 2: Nesse exemplo, existe um combo com várias opções(componente <h:selectOneMenu>). Tais opções são redenrizados pela tag <f:selectItems> e seus valores são obtidos atráves de uma classe. Quando é selecionado o primeiro valor do combo, é desabilitado o componente input text selone_menu_subcolors, nos outros casos o componente é habilitado.

```
<h:selectOneMenu id="selone_menu_colors" value="red">
        <f:selectItems value="#{jsListenerSample.listaCores}"/>
        <t:jsValueChangeListener for="selone_menu_subcolors"
        expressionValue="alert($srcElem.selectedIndex);$srcElem.selectedIndex == 0 ? $destElem.disabled = true :
$destElem.disabled = false" />
</h:selectOneMenu>
<h:inputText id="selone_menu_subcolors" />
```

CONCLUSÃO

Esse tutorial teve como objetivo demonstrar os principais componentes JSF implementados pelo projeto MyFaces Jakarta, demonstrando de forma sucinta a utilização de tais componentes.

Não foram abordados aqui alguns componentes por serem menos importantes.

Maiores informações sobre o projeto MyFaces Jakarta, podem ser obtidos através do site http://myfaces.apache.org/ ou no http://wiki.apache.org/myfaces/MyFacesComponents.