DevMedia - Leitor Digital

Publicação: Java Magazine ed. 85
Artigo: Customizando Componentes no JSF 2.0

[Java Magazine 85 - Índice]

Customizando Componentes no JSF 2.0
Criação de componentes de interface com usuário

José Alexandre Macedo é Mestrando em Informática na Universidade Federal do Espírito Santo. Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.
De que se trata o artigo:
Utilização do JavaServer Faces 2 para criação de componentes de interface com o usuário customizados. Neste artigo você verá o que existe por trás dos componentes customizados, além de aprender a criar seus próprios componentes.

Para que serve:
O JavaServer Faces é uma tecnologia component-based que permite a customização de componentes existentes ou a criação de novos. A construção de componentes IU customizados permite o reuso dos códigos de interface que possuem comportamento. Com isso, é possível aumentar a velocidade e a qualidade do desenvolvimento web através do uso e da construção de bibliotecas de componentes.

Em que situação o tema é útil:
O reuso é uma palavra fundamental no desenvolvimento de software, e diversos paradigmas de desenvolvimento se preocupam com ela há décadas. A customização de componentes IU é útil para as empresas e desenvolvedores que se preocupam com o reuso e que implementam, repetidas vezes, interfaces com o usuário semelhantes que podem ser unificadas em um componente com configurações específicas.

Customizando Componentes no JSF 2.0:
A tecnologia JSF permite a customização de componentes de interface com usuário que pode ser realizada com ou sem o uso de código Java, utilizando, respectivamente, composite components ou non-composite components. Este artigo aborda o uso de composite components para a criação de componentes IU. Além disso, é visto como a arquitetura dos componentes é definida, tendo como base a interface UIComponent, suas subclasses e diversas interfaces importantes. Em seguida são apresentadas as responsabilidades destas classes e interfaces. Na parte prática, o exemplo para construção de componentes IU se dá em cima de um leitor para Twitter, que é apresentado em dois momentos distintos: com e sem comportamento e AJAX. Por fim, é descrito uma forma de agilizar o desenvolvimento de composite components utilizando o assistente de criação disponível na IDE NetBeans.

O JavaServer Faces (JSF) é uma especificação da Sun para desenvolvimento web que utiliza como base componentes. O uso destes componentes para a construção de interfaces com o usuário (IU), simplifica o desenvolvimento mantendo escondida a implementação de tarefas complexas e apresentando ao programador apenas tags com atributos que descrevem as entradas necessárias para o correto funcionamento do componente.
A possibilidade de customização de componentes é um recurso poderoso de extensão para atender às demandas específicas de reuso presentes nas aplicações web atuais. Estas aplicações possuem cada vez mais objetivos em comum (com comportamentos diversificados), que podem vir a se tornar reusáveis de forma fácil com o apoio de JSF. Por exemplo, é comum termos uma tela de login nos sistemas, entretanto cada tela pode ter configurações específicas como a ação necessária para efetuar o login. Dessa forma, ao mudarmos a configuração de uma tela alteramos o seu comportamento.
A partir da versão 2 do JSF, foi definido uma nova maneira para criar componentes IU denominada *composite component*. Esta técnica facilitou a tarefa de customização, dispensando o uso de código Java e definindo um conjunto de tags que permitem construir componentes através de páginas XHTML.

Com isso, o desenvolvedor front-end não precisa mais saber Java para reusar códigos com comportamento, bastando conhecer as tags do JSF. Como o JSF é baseado em componentes que podem ser criados e estendidos, ao longo do tempo surgiram diversas bibliotecas, algumas gratuitas, outras proprietárias. Estas bibliotecas são mantidas por empresas ou comunidades que buscam facilitar a vida do desenvolvedor, com códigos de qualidade e prontos para serem usados nos mais diversos tipos de aplicação web. Alguns exemplos de bibliotecas são RichFaces, PrimeFaces, IceFaces, Tomahawk, OpenFaces, entre muitas outras. Neste artigo, você irá aprender a criar componentes IU customizados utilizando composite components através do exemplo de um leitor para Twitter. Além disso, irá compreender como utilizar o assistente da IDE NetBeans para apoio à customização, tornando ainda mais fácil esta tarefa. Veremos como é simples e útil aplicar esta técnica do JSF 2. Os conceitos serão apresentados aos poucos e o exemplo será explicado em duas etapas para facilitar a compreensão.

O que são componentes de interface com o usuário? O conceito de componente pode ser definido como uma unidade de software utilizada para a construção de vários sistemas que tem como principal característica a possibilidade de configuração para a modificação do seu funcionamento, alterando, dessa forma, o seu comportamento.
Através de uma abstração da implementação podemos realizar substituições destes componentes por outros mais robustos sem muito esforço, desde que estes tenham a mesma especificação (interface). Ao desenvolver um componente é importante definir estas interfaces com cuidado, porque os serviços são providos através delas e para que no futuro não apareçam necessidades que não foram planejadas.
Os componentes IU do JSF são caracterizados por auxiliarem o desenvolvimento de elementos visuais com comportamento que pode ser alterado de acordo com a necessidade. Estes elementos podem ser reutilizados em vários contextos diferentes e estendidos para a composição de novos componentes.
Ao utilizar um componente IU, o desenvolvedor não precisa conhecer a implementação, ele precisa conhecer apenas a interface (contrato). Esta interface informa os atributos necessários (obrigatórios e não obrigatórios) para a utilização do componente.

O JSF oferece diversos componentes básicos para o desenvolvimento de aplicações web que utilizam HTML, entre eles, botões, caixas de texto, formulários, checkboxes, comboboxes e muitos outros. Contudo, durante o desenvolvimento, é comum surgir a necessidade de outros componentes mais avançados que não existam na biblioteca padrão. Nestes casos, o desenvolvedor pode criar os seus próprios componentes. O uso de componentes no JSF tem como objetivo trazer a forma de criar programas desktops para o desenvolvimento web, buscando simplificar este desenvolvimento que é bem mais complexo por envolver diversas outras variáveis, como: request, response e escopo. Um exemplo de simplificação presente em todos os componentes IU está na comunicação entre a camada de visão (páginas) e a camada de controle, que é realizada através da chamada de managed beans nos atributos dos componentes.
Quando customizar componentes IU? A customização de componentes deve ser realizada considerando alguns critérios que justifiquem sua implementação. Por isso, avalie os seguintes critérios antes de

desenvolver seus próprios componentes:

- O componente IU pode existir, então pesquise em fontes como o site JSFCentral (confira o endereço na seção Links), que contém uma lista com mais de 50 bibliotecas e componentes individuais disponíveis, e busque também informações em revistas e livros. Assim você não corre o risco de refazer o componente, economizando tempo e muitas vezes encontrando customizações de ótima qualidade;
- Veja se é realmente necessário um novo componente. Se o componente idealizado não apresentar necessidade de mudança de comportamento, talvez não seja necessário criá-lo. Caso o objetivo seja apenas reaproveitar partes da página que se repetem, como por exemplo um menu, um cabeçalho ou um rodapé, então os templates do Facelets, também disponível no JSF 2, podem resolver o problema;
- Os componentes IU disponíveis no JSF 2 podem ser personalizados através de validadores e conversores. Eles evitam que entradas inválidas cheguem até a lógica de negócio através da validação dos dados e da conversão para tipos primitivos ou objetos Java, respectivamente. Veja se suas necessidades não são atendidas com o uso dos tratamentos de dados, evitando assim a criação de componentes customizados desnecessários.

Tipos de construção para components IU no JSF No JSF existem duas formas de criar componentes customizados, através de composite components ou non-composite components. Vejamos as diferenças entre estas técnicas:
- Composite Components: Esta técnica foi disponibilizada a partir do JSF 2 e dispensa o uso de classes para criar um componente customizado. Nela os componentes são definidos em páginas XHTML que utilizam a biblioteca composite para customização. Essa biblioteca contém diversas tags que definem interface, atributos da interface, implementações, entre outros. Além disso, é possível o tratamento de eventos e listeners através de composite components;
- Non-composite Components: Esta técnica já estava disponível antes da versão 2 do JSF e se diferencia pela necessidade de utilizar classes Java para a criação dos componentes. Este tipo de desenvolvimento pode gerar customizações da mesma forma que composite components. Entretanto, com ela é necessário que o designer saiba programar em Java.

Arquitetura dos componentes IU

Os componentes padrões do JSF podem ser classificados basicamente em duas categorias: aqueles que inicializam uma ação como, por exemplo, um botão, e aqueles que fornecem dados, como um campo de entrada.
Como já foi informado, a característica mais importante dos componentes é a possibilidade da mudança de comportamento. Com JSF o comportamento é definido através do uso dos atributos da interface na implementação, o que é feito durante o processo de construção do componente.
Mesmo desenvolvendo utilizando a técnica *composite components*, é importante entendermos as principais classes e interfaces que estão por trás dos componentes. Por exemplo, para iniciar uma ação, é necessário implementar a interface ActionSource2, onde estará localizada a lógica relacionada ao clique do botão.
O uso das interfaces proporciona uma maior abstração e torna mais fácil a compreensão dos objetivos de cada componente IU. A **Figura 1** apresenta o diagrama de classes resumido das principais interfaces e classes.
De uma forma geral, um componente estende a classe UIComponentBase ou uma das suas subclasses existentes. UIComponentBase implementa a interface UIComponent, e toda classe que implementa esta interface é considerada um componente. O uso dela diminui o acoplamento e torna a estrutura mais flexível para extensões. As demais interfaces são específicas para determinadas responsabilidades e são implementadas pelos componentes de acordo com os objetivos destes.
Conhecer melhor como estas classes e interfaces funcionam é importante para entender no que são transformados os componentes que desenvolvemos. Além disso, caso apareça a necessidade de customização com *non-composite components* você já terá uma noção. A **Tabela 1** apresenta as responsabilidades das classes e interfaces envolvidas na construção de componentes JSF.
Existem diversas outras interfaces envolvidas na construção de componentes, entretanto, como não

vamos trabalhar diretamente com elas, não as veremos. Caso exista o interesse em conhecê-las, consulte a documentação do JSF através da seção **Links** no final do artigo.

[abrir imagem em janela]

Nome	Definição	Responsabilidade	Quem utiliza?
EditableValueHolder	Interface	Específica para componentes editáveis. Inclui suporte a eventos e validadores relacionados.	**UIInput, UISelectBoolean, UISelectMany, UISelectOne** e subclasses destes.
ActionSource2	Interface	Faz com que um **ActionEvent** seja disparado quando o usuário clica no componente.	**UICommand** e suas subclasses.
PartialStateHolder	Interface	Salva o estado dos objetos do componente entre os *requests*.	Todos os **UIComponent** e suas subclasses.
ValueHolder	Interface	Possui um valor que não pode ser editado pelo usuário.	**UIOutput** e suas subclasses.
NamingContainer	Interface	Garante que os componentes declarados dentro dele tenham ids únicos.	**UIForm, UINamingContainer, UIData** e as subclasses destes
ClientBehaviorHolder	Interface	Permite atribuir conteúdos de *scripts* para eventos *client-side*.	Todos os componentes que redefinem os elementos HTML.
UIComponent	Interface	Interface base para todos os componentes IU do JSF.	Todos.
UIComponentBase	Classe Abstrata	Classe que fornece implementação concreta para muitos métodos básicos e comuns aos componentes.	Todos.

Tabela 1. Definição de interfaces e classes utilizadas pelos componentes IU

[abrir imagem em janela]

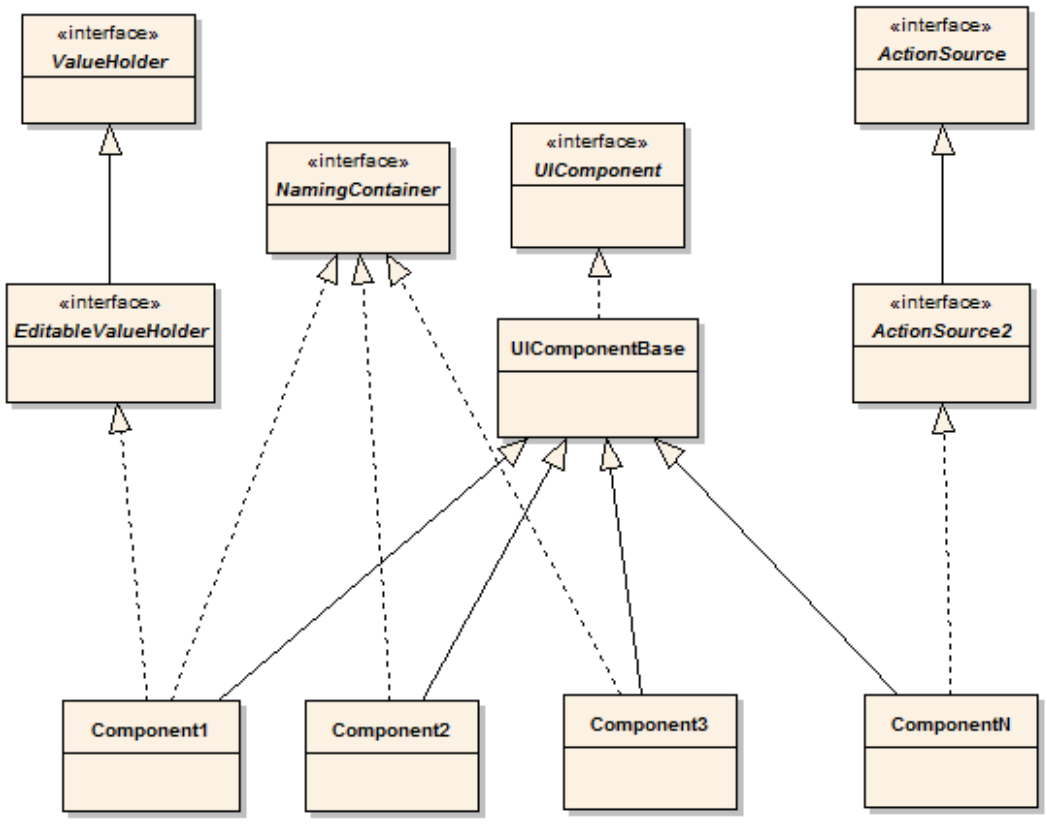

Figura 1.

Classes e interfaces por trás dos componentes IU

Criando componentes IU utilizando composite components Com o objetivo de tornar mais fácil o aprendizado da customização de componentes, vamos criar um leitor para Twitter personalizado. Este leitor permitirá visualizar os últimos tweets de um usuário e utilizará AJAX nativo do JSF 2 para atualizar as mensagens e trocar o usuário. Este exemplo foi escolhido por ser simples de implementar e útil para o

desenvolvedor.
Os pontos definidos no tópico “Quando customizar componentes IU?” foram levados em consideração na escolha do leitor customizado. Não foram encontrados componentes com a mesma funcionalidade, não é possível utilizar templates para resolver o problema e conversores e validadores não são capazes de gerar o leitor para Twitter.
Aprenderemos a criar o leitor customizado em duas etapas: na primeira exibindo os tweets de um usuário específico, porém sem permitir trocar o usuário, exibir tweets novos e mudar o tamanho do componente, e na segunda permitindo.
O exemplo foi desenvolvido na IDE NetBeans. Entretanto, você pode utilizar a IDE de sua preferência para customizar componentes. A **Figura 2** ilustra como ficará no final da primeira etapa o leitor para Twitter.

Esta página é importante pois é nela que o nosso componente customizado é declarado e requisitado, ou seja, é nela que o componente será renderizado para o usuário. A Listagem 1 descreve a página index.xhtml. O primeiro trecho que merece destaque é a declaração da nossa biblioteca de componentes customizados, denominada minhaBibliotecaComponentes. Esta declaração é realizada através do trecho: xmlns:jam="http://java.sun.com/jsf/composite/ minhaBibliotecaComponentes" O caminho padrão para as bibliotecas de componentes customizados engloba http://java.sun.com/jsf/composite/NOME_DA_BIBLIOTECA. Uma biblioteca de composite components nada mais é do que uma pasta que contém uma página XHTML para cada componente. O NOME_DA_BIBLIOTECA é o mesmo nome da pasta. O sufixo jam é utilizado para chamar os componentes da nossa biblioteca na página index.xhtml. Lembrando que é interessante sempre utilizar sufixos pequenos para facilitar a chamada dos componentes.

[abrir imagem em janela]

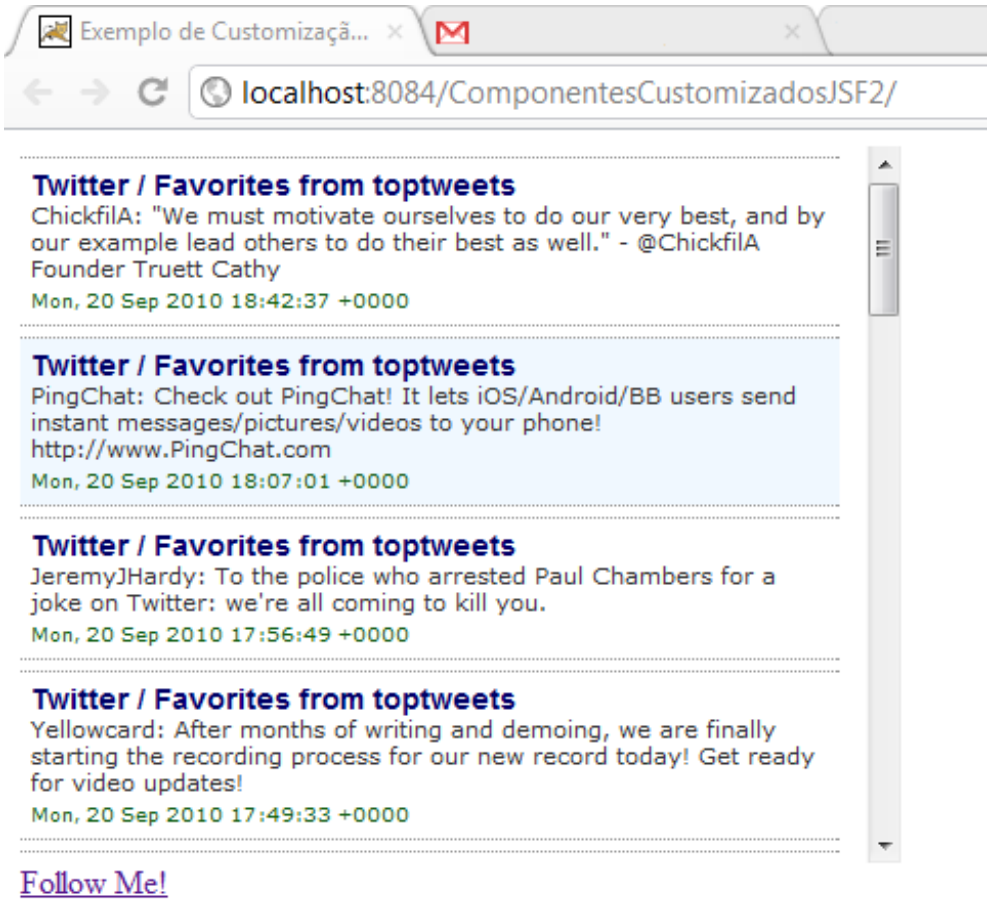

Figura 2.

Visualização do componente customizado na primeira etapa

Listagem 1. index.xhtml: Página com a chamada ao componente IU customizado<?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://java.sun.com/jsf/html"
   xmlns:jam="http://java.sun.com/jsf/
    composite/minhaBibliotecaComponentes">


   Componentes</title>


   <jam:tweets/>

</html>
```

Para que a biblioteca seja reconhecida quando for declarada, a pasta *minhaBibliotecaComponentes* deve estar dentro da pasta *resources* (que também precisa ser criada), e que por sua vez deve estar localizada na pasta *web*, como apresentado na **Figura 3**. Fazendo isso o projeto estará pronto para utilizar os componentes customizados que estiverem dentro da nossa biblioteca.
Continuando a análise da página *index.xhtml*, temos a chamada do nosso leitor para Twitter sendo realizada no seguinte trecho:
O nome tweets é o mesmo nome da página onde o componente é descrito, ou seja, *tweets.xhtml*. Deste modo, toda vez que surgir a necessidade em uma página de visualizar os últimos tweets de determinado usuário, o desenvolvedor pode utilizar esta simples chamada ao componente. Com isso, economizamos tempo e diversas linhas de código depois que o componente está criado.
Estes são os pontos importantes da página *index.xhtml*, responsável por renderizar o nosso componente customizado. Vamos conhecer agora, através da **Listagem 2**, os códigos e tags que devem ser implementados para gerar o componente tweet.
A análise desta listagem permitirá entender como a customização de componentes funciona, e em seguida veremos as classes e arquivos que auxiliam na busca dos Tweets e na estilização do componente.
Existem duas tags básicas para a construção de componentes, a interface e a implementation.

Estas tags pertencem à biblioteca composite, declarada através da chamada:
xmlns:cc="http://java.sun.com/jsf/composite" A interface é representada na página com o sufixo cc por , e é responsável por definir o contrato de uso do componente. Através dela definimos as configurações do componente que podem ser alteradas. Como nesta primeira parte não teremos configurações, não existem atributos dentro da interface. A tag implementation, representada por , é responsável por definir o conteúdo que será renderizado quando o componente for utilizado. Na primeira etapa esta tag irá listar os últimos tweets de um usuário, e na segunda utilizará atributos definidos na tag interface para alterar o comportamento do componente. Os acessos aos atributos da interface na implementation são realizados através de uma variável específica, explicada na segunda etapa do nosso exemplo.

O conteúdo do componente possui a definição do css através da tag . Este css permite renderizar o componente com uma aparência melhor, através da folha de estilos do arquivo estilos.css. Para isso, é necessário criar uma pasta chamada css, dentro da pasta resources, onde estilos.css deverá ficar. Este arquivo deve conter a folha de estilos igual a da Listagem 3.

Listagem 2. tweets.xhtml: Página com a descrição do composite component tweets.<?xml version="1.0" encoding="UTF-8"?>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

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



    name="estilo.css"/>




          value="#{tweetMB.tweets}"
        var="tweet">


            target="_blank">
              #{tweetMB.usuario.nome}
              #{tweet.descricao}
              #{tweet.data}
            </a>



    </div>

    target="_blank">Follow Me!
  </cc:implementation>
</html>
```

Listagem 3. estilos.css: Folha de estilos do componente tweets

```
#links {
    height:350px;
        width:430px;
        overflow:auto;
}
#links ul {
    list-style-type:none;
    margin:0;
    padding:0;
    width:400px;
}
#links li {
    border:1px dotted #999;
    border-width:1px 0;
    margin:5px 0;
}
#links li a {
    color:#000066;
    display:block;
    font:bold 14px Arial, Helvetica, sans-serif;
    padding:5px;
    text-decoration:none;
}

* html #links li a {width:400px;}
/*Necessário para que funcione no IE6*/
#links li a:hover {
    background-color:#F0F8FF;
}
#links a em {
    color:#333;
    display:block;
    font:normal 11px Verdana, Arial, Helvetica, sans-serif;
    line-height:125%;
}
#links a span {
    color:#125F15;
    font:normal 9px Verdana, Arial, Helvetica, sans-serif;
    line-height:150%;
}
```

[abrir imagem em janela]

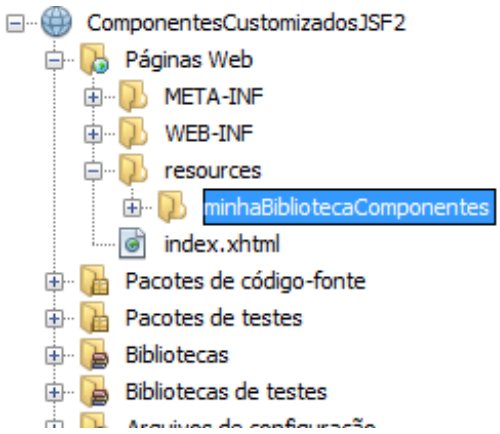

Figura 3.

Estrutura de arquivos para a biblioteca customizada

Na sequência, a implementação do componente utiliza uma repetição através da tag para apresentar na tela os últimos tweets de um usuário. Os tweets são carregados através de uma lista de Tweets presente no *managed bean* TweetMB (**Listagem 4**). Este *managed bean* é usado também para ter acesso a outras informações como nome e link do usuário. Os métodos carregarTweets() e carregarUsuario() utilizam as

variáveis username e caminhoFeed para buscar o *feed* dos últimos tweets. Sendo que, username é o nome de usuário do Twitter e caminhoFeed descreve o caminho básico da url que permite acesso ao *feed* de um usuário. Os tweets buscados ficam armazenados na Collection tweets.
Toda vez que o método setUsername() é chamado, os métodos carregarTweets() e carregarUsuario() também são, tendo como objetivo atualizar as variáveis tweets e usuario. Por fim, o construtor configura o username para “toptweets”, sendo os tweets deste usuário exibidos ao fim desta primeira etapa.
Como pode ser observado, foram utilizados neste *managed bean* dois *beans* auxiliares: Tweet e User, descritos na **Listagem 5** e **Listagem 6**, respectivamente. Estes *beans* também devem ser implementados.
Neste momento nosso componente está pronto para funcionar carregando os tweets do usuário toptweets.
A partir de agora iremos iniciar a segunda parte, realizando algumas alterações que permitirão a configuração do componente e a utilização de AJAX para atualizar tweets e trocar o usuário.
Evoluindo nosso leitor de Tweets Nosso componente já funciona, mas ainda não pode ser configurado. Nesta segunda etapa você irá conhecer e entender as tags assim como os conceitos para torná-lo configurável e permitir o uso de AJAX.
A configuração de um componente é possível através do uso da tag <cc:attribute> dentro da tag <cc:interface>. Cada attribute permite a passagem de um parâmetro para ser utilizado na implementação do componente. Esta tag possui diversas configurações possíveis. Vejamos a lista de algumas delas:
- name: define o nome do atributo. Este nome serve para chamar o atributo na implementação do componente;
- default: define um valor padrão para o atributo. Assim, quando o desenvolvedor não utilizar este atributo na chamada da tag, o valor default é usado;
- required: torna obrigatório o preenchimento do atributo quando o componente for utilizado;
- type: define um tipo para o atributo que deve ser compatível com os tipos permitidos na linguagem Java;

- method-signature: especifica que o valor do atributo será um método descrito por uma assinatura. Este atributo não pode ser usado junto ao atributo type pois eles são mutuamente exclusivos. Caso sejam usados juntos, method-signature é ignorado;
- targets: utilizado para definir os alvos do atributo. Os alvos são especificados através dos ids dos componentes que estão em implementation;
- displayName: utilizado para descrever o atributo;
- shortDescription: uma descrição curta do atributo.

Listagem 4. TweetMB.java: Managed Bean ligado ao componente tweet

```
import com.sun.org.apache.xerces.internal.parsers.DOMParser;
import java.util.ArrayList;
import java.util.Collection;
import javax.faces.bean.ManagedBean;
import javax.faces.event.ActionEvent;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

@ManagedBean
public class TweetMB {

  private User usuario = new User();
  private Collection tweets =
   new ArrayList();
  private static final String
        caminhoFeed = "http://twitter.com/statuses/
        user_timeline.rss?screen_name=";
  private String username;
```

```
public TweetMB() {
  this.setUsername("toptweets");
}

public void setUsername(String username) {
  this.username = username;
  carregarTweets(null);
  carregarUsuario(null);
}

public void carregarTweets(ActionEvent ev) {
  if (username != null) {
    try {
      tweets.removeAll(tweets);
      DOMParser parser = new DOMParser();
      parser.parse(caminhoFeed + username);
      Document doc = parser.getDocument();
        Tweet tweet = new Tweet();
        NodeList description =
        doc.getElementsByTagName("description");
        NodeList pubDate = doc.
        getElementsByTagName("pubDate");
        NodeList link = doc.
        getElementsByTagName("link");
        for (int i = 1; i < description.
        getLength(); i++) {
        tweet = new Tweet();
        tweet.setDescricao(description.item(i).
        getTextContent());
        tweet.setData(pubDate.item(i).
        getTextContent());
        tweet.setLink(link.item(i).
        getTextContent());
        tweets.add(tweet);
      }
    } catch (Exception ex) { /* tratar */ }
  }
}

public void carregarUsuario(ActionEvent ev) {
  if (username != null) {
    try {
      DOMParser parser = new DOMParser();
      parser.parse(caminhoFeed + username);
      Document doc = parser.getDocument();
      usuario.setNome(doc.
      getElementsByTagName("title").
       item(0).getTextContent());
      usuario.setLink(doc.
      getElementsByTagName("link").
      item(0).getTextContent());
    } catch (Exception ex) { /* tartar */ }
  }
}
```

```
  //getters and setters

}
```

Listagem 5. Tweet.java: Classe bean auxiliar para listagem de tweets

```
public class Tweet {

  private String descricao;
  private String data;
  private String link;

  public Tweet() { }

  //getters and setters
}
```

Listagem 6. User.java: Classe bean auxiliar para obter informações do usuário

```
public class User {

  private String nome;
  private String link;

  public User() { }

  //getters and setters
}
```

O conhecimento destes atributos da tag attribute permite que o desenvolvedor saiba o que pode utilizar quando for criar outros componentes customizados.
Outro ponto de destaque nesta segunda etapa é o uso do AJAX nativo do JSF 2 através da tag <f:ajax>. Ela é usada dentro da tag de botões e links e permite uma nova renderização de componentes da página. Para saber quais deles devem ser novamente renderizados basta informar seus ids, separando-os com vírgula, no atributo render da tag <f:ajax>. Você também pode especificar o @form no render para que todo o formulário seja renderizado, como foi feito em nosso exemplo.
As mudanças necessárias para o componente passar a utilizar AJAX e se tornar configurável são apresentadas na **Listagem 7**. As partes em negrito do código são as modificações necessárias.
O nosso componente passou a ter três atributos para configurá-lo. O primeiro, obrigatório, chamado managedBean é usado para passar o *managed bean* que realiza as buscas pelos tweets. O segundo, chamado largura possui um valor padrão 300 e é utilizado para definir a largura do leitor de tweets. O terceiro e último, chamado altura, tem um valor padrão de 400 e define a altura do componente.
A tag implementation passou a ter uma tag <h:form>, necessária para que ao clicar nos botões de alteração de usuário e atualização de tweets, os dados para a nova renderização sejam enviados ao servidor e as alterações na página executadas.

A tag <f:ajax>, usada nos dois botões, utiliza os atributos execute e render. O primeiro, execute, contém o id usernameId, que é usado para atualizar os dados no servidor do componente que tem este id. O segundo, render, contém o @form, utilizado para informar que todo o formulário deve ser novamente renderizado.
A utilização dos atributos definidos na tag interface é feita com o uso de uma variável pré-definida: #{cc.FUNC}, onde FUNC são as funcionalidades que auxiliam o acesso aos atributos. Essa variável é extremamente importante, pois permite a interação do componente customizado com as informações

inseridas pelo desenvolvedor na hora do uso do componente.
Vejamos algumas funcionalidades permitidas para esta variável:
- #{cc.attrs.NOME_ATRIBUTO}: Podemos utilizar o attrs para capturar o valor do atributo, onde NOME_ATRIBUTO é definido na tag atributte dentro da interface. Temos vários exemplos no nosso componente customizado, um deles pode ser visto na Figura 4;

- #{cc.clientId}: O clienteId é utilizado para localizar a verdadeira identificação do elemento. Um exemplo seria #{cc.clientId}:usernameId;
- #{cc.parent}: Quando o componente customizado tiver sido criado dentro de outro componente, o parent traz a possibilidade de manipular a variável #{cc.FUNC} do componente pai;
- #{cc.children}: Caso o componente customizado tenha um componente filho, é possível manipular a variável #{cc.FUNC} do filho utilizando o children.

Listagem 7. tweets.xhtml: Página do composite component tweets modificada para utilizar AJAX e se tornar configurável<?xml version="1.0" encoding="UTF-8"?>

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


    required="true"/>




    name="estilo.css"/>



                value="#
                {cc.attrs.managedBean.username}"/>

      ajax execute="usernameId"
      render="@form"/>


     largura}
      px;height: #{cc.attrs.altura}px">



          value="#{cc.attrs.
        managedBean.tweets}"
         var="tweet">


           target="_blank">
            #{cc.attrs.managedBean.usuario.nome}
```

```
        #{tweet.descricao}
        #{tweet.data}
      </a>
```

```
" target="_blank">Follow Me!</a>

actionListener="#
  {cc.attrs.managedBean.carregarTweets}">
  ajax execute="usernameId"
  render="@form"/>
```

```
</html>
```

[abrir imagem em janela]

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:cc="http://java.sun.com/jsf/composite"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core">
    <cc:interface>
        <cc:attribute name="managedBean" required="true"/>
        <cc:attribute name="largura" default="300"/>
        <cc:attribute name="altu  default="400"/>
    </cc:interface>
    <cc:implementation>
        <h:outputStylesheet library="cs  name="estilo.css"/>
        <h:form prependId="false">
            <h:outputText value="Username:"/>  :inputText id="usernameId" value="#{cc.attrs.man
            <h:commandButton value="Alterar">
                <f:ajax execute="usernameId" rend  orm"/>
            </h:commandButton>

            <div id="links" style="width: #{cc.attrs.largura}px;height: #{cc.attrs.altura}px">
            <ul>
                <ui:repeat  value="#{cc.attrs.managedBean.tweets}" var="tweet">
                    <li>
                        <a href="#{tweet.link}" target="_blank">
                            #{cc.attrs.managedBean.usuario.nome}
                            <em>#{tweet.descricao}</em>
                            <span>#{tweet.data}</span>
```

Figura 4.

Exemplo de uso da variável #{cc.attrs.NOME_ATRIBUTO}.

Com a variável #{cc.FUNC}, são criadas várias possibilidades de customização que podem ser exploradas, de acordo com o problema, para a criação de outros componentes customizados. Por exemplo, poderíamos customizar um novo componente leitor de notícias e utilizar o nosso leitor de tweets na tag implementation dele (Listagem 8). Dessa forma, nosso leitor de tweets poderia acessar os atributos do novo componente utilizando a funcionalidade parent da seguinte forma: #{cc.parent.attrs.NOME_ATRIBUTO}, como na Listagem 9. A página index.xhtml também precisa de algumas mudanças, já que um dos atributos criados deve ser, obrigatoriamente, definido no nosso componente. Estas mudanças podem ser vistas na Listagem 10, nas partes em negrito do código. Com isso, podemos verificar que três atributos do nosso componente customizado foram definidos para configurá-lo. O managed bean TweetMB que antes era utilizado diretamente dentro do componente, agora é passado como parâmetro. Os atributos altura e largura também foram alterados para experimentar um tamanho diferente no componente. Com estas mudanças permitimos uma adaptação do leitor de acordo com o local onde ele será exibido. A ligação entre a definição da interface, a implementação e a página index.xhtml que utiliza o componente pode ser vista na Figura 5.

Listagem 8. noticias.xhtml: Um exemplo de uso do nosso leitor de tweets em um novo componente

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




   ...


   ...



</html>
```

Listagem 9. tweets.xhtml: Exemplo de como poderia ser usada a funcionalidade parent no leitor de tweets

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


   ...
   Leitor de Notícias Responsável:

   ...


</html>
```

Listagem 10. index.xhtml: Página onde o componente customizado com configuração é utilizado<?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://java.sun.com/jsf/html"
xmlns:jam="http://java.sun.com/jsf/
composite/minhaBibliotecaComponentes">

Componentes</title>

altura="600" largura="460"/>
</h:body>
</html>

[abrir imagem em janela]

Figura 5.

Ligação entre o componente customizado e a página que o utiliza

O resultado para o usuário final gerado pelo nosso componente customizado pode ser visto na **Figura 6.**

Assistente de criação de componentes IU (NetBeans) A customização de componentes pode ser facilitada com o uso da IDE NetBeans. A partir da versão 6.8, foi disponibilizada uma interface para criação de composite component através de um assistente de forma rápida e fácil. Veja os passos necessários para utilização do assistente.

1º Passo
Selecione o trecho do código que deverá virar um componente IU e clique com o botão direito do mouse no código selecionado. Dentro da opção Refatorar, selecione a opção Converter a Componente Composto (observe a Figura 7).

2º Passo
O assistente será inicializado. O nome do componente e o nome da pasta deverão ser preenchidos, sendo utilizados posteriormente como o nome da página XHTML e o nome para a biblioteca do componente, respectivamente. O nome padrão para a biblioteca é ezcomp. Após completar as informações finalize o assistente (Figura 8).

[abrir imagem em janela]

Figura 6.

Resultado do componente customizado com comportamento

[abrir imagem em janela]

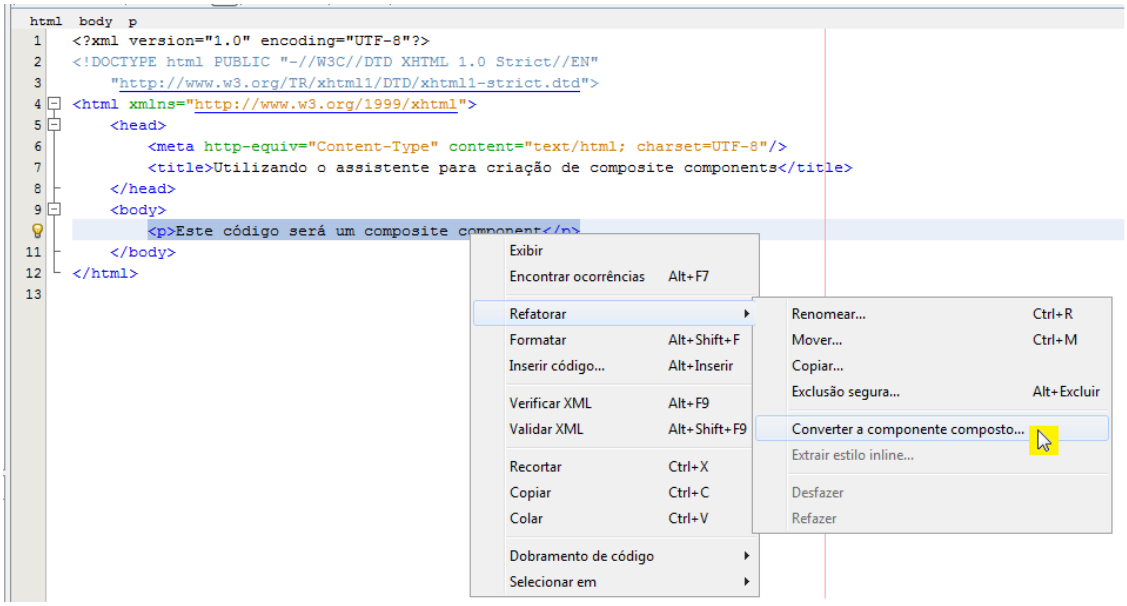

Figura 7.

Seleção do código que irá virar um componente customizado pelo assistente do NetBeans

[abrir imagem em janela]

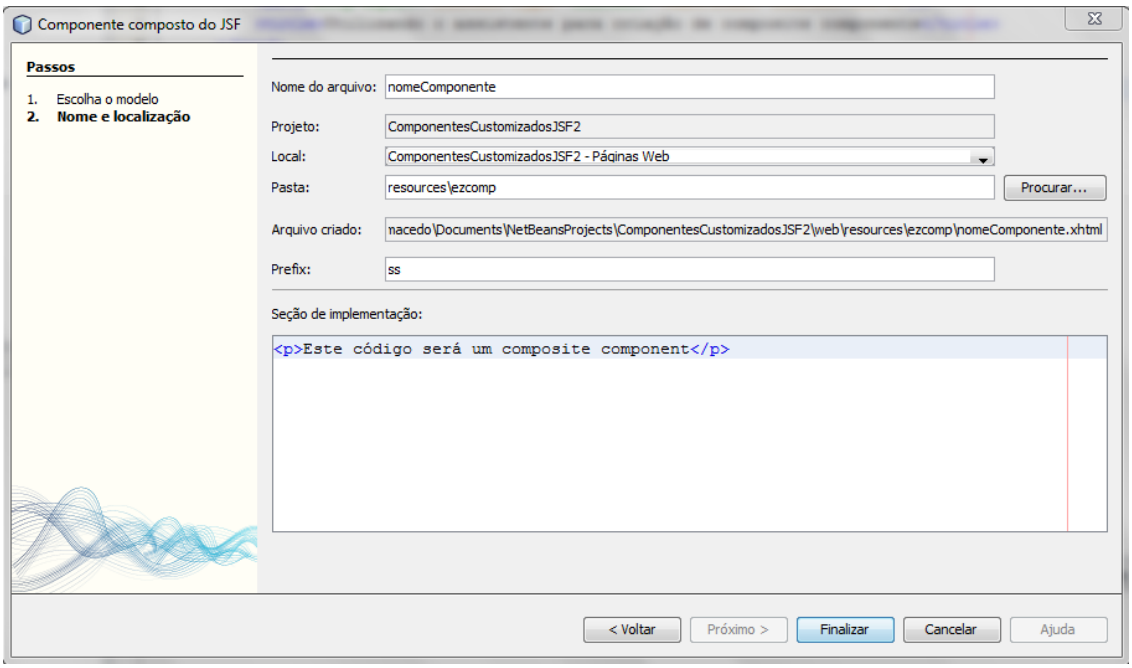

Figura 8.

Assistente para configuração de um novo componente customizado

Ao encerrar o assistente o código do componente customizado será gerado e aparecerá como na **Listagem 11**.
Em seguida a chamada para o novo componente é inserida, automaticamente, na página onde o assistente foi iniciado, e assim a página passa a ter o código de acordo com a **Listagem 12**.
Com a utilização do assistente para criação de componentes a tarefa de customização fica ainda mais simples e rápida, trazendo comodidade ao desenvolvedor.
Na vídeo aula deste artigo demonstraremos na prática a criação de componentes com o JavaServer Faces 2.0.

Listagem 11. nomeComponente.xhtml: Página gerada pelo assistente com o código do componente customizado.<?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:cc="http://java.sun.com/jsf/composite">
  <!-- INTERFACE -->




    <p>Este código será um composite component</p>

</html>
```

Listagem 12. index.xhtml: Página gerada pelo assistente com o código do componente customizado<?xml version="1.0" encoding="UTF-8"?>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ss="http://java.sun.com/jsf/composite/ezcomp">


     charset=UTF-8"/>

     de composite components</title>


    <ss:nomeComponente/>

</html>
```

Conclusão

Este artigo apresentou as técnicas para customização de componentes dando destaque às novidades do JSF 2. Nessa nova versão a customização ficou mais fácil, através da técnica *composite components*, a partir da qual se tornou possível construir novos componentes sem a utilização de código Java. Vimos as tags da biblioteca *composite* necessárias para a customização e as possibilidades de configuração que permitem a criação de uma ampla variedade de componentes para as mais variadas finalidades.
Foi apresentado também um assistente para criação de componentes, que agiliza o processo através da geração de códigos comuns a todos os *composite components*, tornando possível uma customização mais rápida.
A customização pode gerar um ganho na velocidade do desenvolvimento com JSF. A partir do momento em que os componentes começam a ser reutilizados, os desenvolvedores podem se preocupar mais com outras partes críticas da aplicação.

Vídeos

Este artigo não possui comentários. Seja o primeiro a comentar!