Soluções adotadas no SIGEventos para a migração de JSF1.2 para JSF 2.X

Jadson Santos – jadson@info.ufrn.br

Versão 1.5

07/02/2014

O principal problema encontrado quando se migra de JSF 1.2 para JSF 2.0 é que as funcionalidades JSP deixam de existir. Já que por padrão JSF 2.0 usar *fecelets* e páginas xhtml.

O problema é que muitas funcionalidades nas páginas dos sistemas misturam JSF 1.2 com JSP. Principalmente as páginas da arquitetura (cabeçalho.jsp, rodapé.jsp) que fazem uso massivo de*scriplet, scripting , expression e declarations*.

Páginas compartilhadas:

```
<c:import context="/shared"
url="/sistemas.jsp?sistemaAtual=sigEvento" var="sistemas"
scope="request"/>
```

Não funciona. E, até onde eu sei, não há como fazer "import" dinâmico com *facelets*, então crie uma página própria na aplicação, vai duplicar, sempre que um novo sistema for criado, tem que adicionar em todas as páginas de logins dos outros sistemas.

Ps.: Por essas coisas que muita gente fala mal de JSF.

Validação do XML:

```
<br>
```

Não é uma tag xhtml válida, todas as tags têm que ser fechadas agora:

```
<br/>
```

Caracteres como '&', '<' e '>' não são permitido em páginas xhtml

Usar

```
&amp; , &lt;  e  &gt;
```

Funcionalidades de JSP:

```
<c:if test="${ sessionScope.usuario != null}">
```

Não funciona mais, use:

```
<ui:fragment rendered="#{ sessionScope.usuario != null}">
```

```
<c:forEach var="l" items="#{ mbean.lista}">
```

Não funciona mais, use:

```
<ui:repeat var="l" value="#{mbean.lista}">
```

```
class="${ status.index % 2 == 0 ? "linhaPar" : "linhaImpar" }"
```

por

```
class="#{ status.index % 2 == 0 ? 'linhaPar' : 'linhaImpar' }"
```

Uso de variáveis temporárias para destacar colunas em laços:

```
<c:set var="id" value="-1" scope="request"/>
<c:forEach var="dado" items="#{dados}" varStatus="status">
     <c:if test="${ id != dado.id}">
          <%-- imprime a informação e seta o novo valor --%>
          <c:set var="id" value="${dado.id}" scope="request"/>
     </c:if>
</c:forEach>
```

A solução encontrada foi substituir por cálculos *inline* usando a coleção iterada:

```
<ui:repeat var="dado" value="#{dados}" varStatus="status">
     <ui:fragment rendered="#{ status.index == 0
     || dados[status.index].id != dadis[status.index-1].id }">
     </ui:fragment>
</ui:repeat>
```

```
<%@page import="br.ufrn.arq.util.AmbienteUtils"%>

<%=AmbienteUtils.dadosVersao("versao_sigeventos").get("sistema")
%>
```

Não funciona mais, crie um MBean que retorne essas informações sobre o ambiente:

```
public class AmbienteUtilsMBean {

      public String getVersaoSistema() {

            return AmbienteUtils

            .dadosVersao("versao_sigeventos").get("sistema");

      }

}

#{ambienteUtilsMBean.versaoSistema}
```

```
<%= br.ufrn.arq.util.UFRNUtils.getCopyright(2006) %>
```

Não funciona mais, crie um MBean para isso como no caso anterior ou utilize “custom functions” do JSF.

```
<namespace>http://sinfo.ufrn.br</namespace>

<function>

      <function-name>copyright</function-name>

      <function-class>

            br.ufrn..arq.util.CustomFunctions</function-class>

      <function-signature>

            java.lang.String getCopyright(java.lang.String)
```

```
      </function-signature>
</function>
```

Nas páginas JSF:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:ufrn="http://sinfo.ufrn.br">
      ...
      #{ufrn:getCopyright(2006)}
```

```
<%-- Variáveis globais --%>
<c:set var="ctx" value="<%= request.getContextPath() %>"/>
```

Não funciona, crie um MBean com espoco de *application* para guardar essas variáveis

```
public class ApplicationContextMBean{
      public String getContextPath() {
            return ( (HttpServletRequest)
FacesContext.getCurrentInstance().getExternalContext().getReques
t()).getContextPath();
      }
}
#{applicationContextMBean.contextPath}
```

```
<%
boolean sipacAtivo = Sistema.isSipacAtivo();
%>
```

Novamente crie um Mbean para usar nas páginas:

```
public class VerificacaoSistemaMBean {
      public boolean isSipacAtivo() {
            return Sistema.isSipacAtivo ();
      }
}
#{verificacaoSistemaMBean.isSipacAtivo}
```

Tags da UFRN:

```
<ufrn:checkRole papeis="<%= new int[]{Papeis.ADMINISTRADOR} %>">
```

Não funciona mais, novamente crie um MBean com um método para cada papel do sistema:

```
public class SigEventosPapeisMBean extends AbstractController{

      public boolean isUserAdministratorSigEventos(){
            return isUserInRole(SIGEventoPapeis
                  .ADMINISTRADOR_SIGEVENTOS);
      }
}
<ui:fragment
rendered="#{sigEventosPapeisMBean.userAdministratorSigEventos}">

<ufrn:subSistema/>
```

Algumas taglibs dá para substituir por custom tag estido JSF 2.0 usando <ui:componente>

xmlns:ufrn=http://sinfo.ufrn.br

subsistema.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>

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

<html xmlns="http://www.w3.org/1999/xhtml"

      xmlns:h="http://java.sun.com/jsf/html"

      xmlns:ui="http://java.sun.com/jsf/facelets">

    <ui:component>

         <b> <h:outputText
value="#{customComponentsMBean.linkSubSistema}" escape="false"
/> </b>

    </ui:component>

</html>
```

```
<ufrn:help>

Texto

</ufrn:help>
```

Usar o novo UFRN Help usando JSF 2.0 e primefaces para criar o tooltip:

```
<ufrn:help id=”1” text="Texto" width="500px;" />
```

```
<ufrn:format type="data" valor="${mBean.obj.dataInicio}"/>
```

Usa o modo JSF 2.0 de ser:

```
#{mBean.obj.dataInicioFormatada}
```

Algumas coisas deixaram de existir:

```
<a4j:support event="onchange" reRender="formLogin" />
```

Não existe mais, usar:

```
<p:ajax event="change" update=" formLogin " />
```

```
<a4j:outputPanel ajaxRendered=”true” />
```

Não existe mais, usar:

```
<p:fragment autoUpdate="true">
```

```
<a4j:keepAlive beanName="mBean"/>
```

Não existe mais no RichFaces 4.3, usar

@ViewScoped nos seus MBeans JSF 2.X se o caso de uso for em apenas 1 página

OU

Utilizar o CustomScope para casos de uso que usam várias páginas

Esse tutorial mostra como criar um custom scope:

http://blog.oio.de/2012/07/24/jsf-2-custom-scopes-without-3rd-party-libraries/

Como usar CustomScope ?

Um escopo onde são armazenados os MBean em JSF é nada mais que um mapa cujas chaves são os nomes do MBean e os valores os próprios MBean.

Para criar um CustomScope deve-se criar uma mapa com o nome do escopo adiciona-lo ao FacesContext do JSF e utilizar um Mbean de sessão para criar e destruir esse escopo assim que ele não for mais necessário.

Assim os dados dos Mbean não precisam mais ficar o tempo todo na memória do servidor ocupando espaço enquanto o usuário estiver logado.

O MBean em session criado é só para ter um Mbean que dure mais que o customScope para poder gerencia-lo, contudo ele não armazenará os dados do custom scope, então não ocupará muito espaço na memória do servidor.

Uma boa maneira de gerenciar o custom scope criado é sempre ao iniciar algum caso de uso no menu principal, iniciar o custom scope com um listener:

```
<h:commandButton value="start" action="#{exampleBean.tela1}">

     <f:actionListener
     type="br.ufrn.sigevento.arq.jsf.CreateConversationScope" />

</h:commandButton>
```

Para destruir colocar a chamada

```
#{ conversationScopeMBean.destroyCustomScope}
```

no menu principal da aplicação menu.xhtml, assim os dados permanecerão em memória durante todo o caso de uso, e sempre que o usuário terminar o caso de uso e voltar para o menu da aplicação, eles serão apagados.

Ou então finalizar manualmente na última ação do caso de uso:

```
<h:commandButton value="Finalizar" action="menu.xhtml">

     <f:actionListener

          type="br.ufrn.sigevento.arq.jsf.DestroyCustomScope"
     />

 </h:commandButton>
```

Custom Scope oferece uma maneira melhor de gerenciar os dados em memória da aplicação do que ter que se lembrar de colocar um keepAlive em todas as páginas chamadas pelo caso de uso como era feito antes.

Injeção de Dependências:

A injeção de dependência no JEE6 aparentemente não funciona no JBoss 5.1. Sendo assim para usar o @CustomScope não será possível usar as anotações:

```
@ManagedBean

@CustomScoped("#{CONVERSATION_SCOPE}")
```

As anotações do Spring usadas até agora também não gerenciam o scope do JSF 2.x corretamente.

```
@Component("mBean")

@Scope("request")
```

Em vez disso vamos ter de usar as velhas configurações em arquivos XML, talvez quando mudamos para JBOSS 6 ou 7 possamos usar anotações.

```
<managed-bean>

<managed-bean-name>exampleBean</managed-bean-name>

<managed-bean-
class>br.ufrn.sigevento.arq.jsf.ExampleBean</managed-bean-class>

<managed-bean-scope>#{CONVERSATION_SCOPE}</managed-bean-
scope>

</managed-bean>
```

JavaScript

```
getEL('formBuscaPadraoPessoas:checkLogin').dom.checked = true;
```

substituir por:

```
$('.myCheckbox').prop('checked', true); // Jquery like
```

Novo esquema de mensagens:

A além das alterações citas acima, foi preciso fazer pequenas adaptações, como uma mudança no estilo das mensagens mostradas aos usuários na aplicação.

Usam o esquema de mensagens do primefaces.

```
/**
*Override the architecture default message to use the primefaces
* message, because it is more beautiful and work with ajax
*/
@Override
public void addMensagemInformation(String mensagem) {
	FacesContext.getCurrentInstance().addMessage("normal"
	, new FacesMessage(FacesMessage.SEVERITY_INFO,
	"Informação", mensagem));
	erros = new ListaMensagens();
}
```

```
/**
* A new type of message:  use when the page is very big, to user
* doesn't need page up to see the message
*/
public void addMensagemInformationFlutuante(String mensagem) {
      addMensagemInformation(mensagem);
      FacesContext.getCurrentInstance().addMessage("flutuante",
       new FacesMessage(FacesMessage.SEVERITY_INFO, "Informação",
      mensagem));
      erros = new ListaMensagens();
}
```

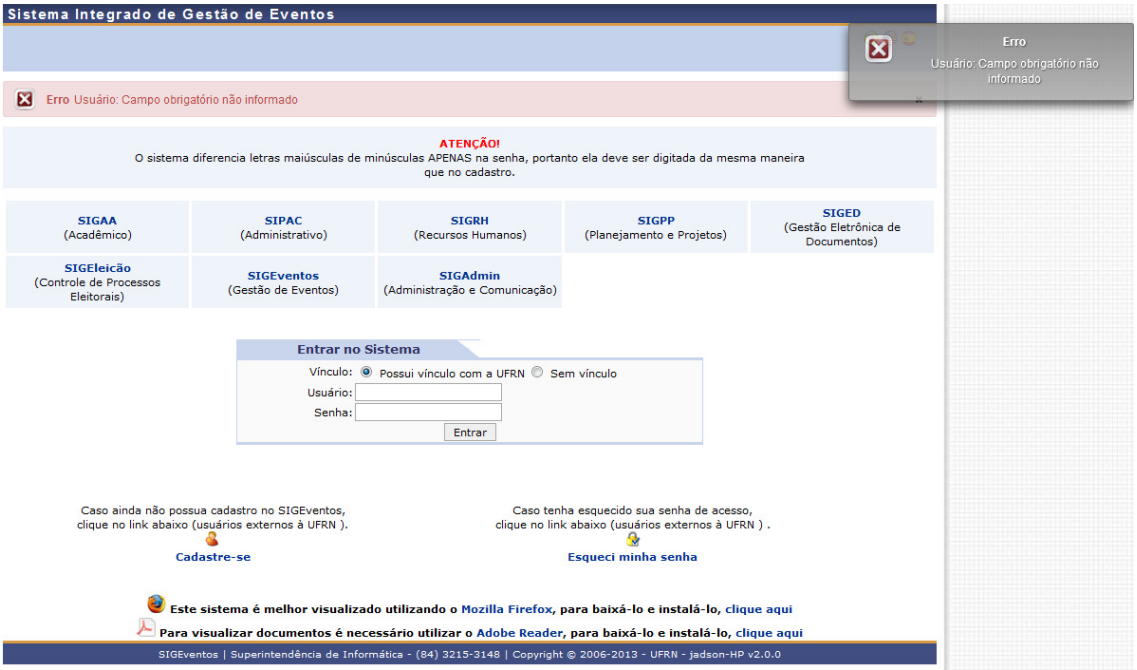
```
<div id="mensagem">
<!--  Mensages in the top of the page -->
<p:messages id="messages" showDetail="true" autoUpdate="true"
closable="true" for="normal"/>

<!--  Mensages the keep 5s in the same place when you roll the
page (use when the page is very big, to user doesn't need page
up to see the message ) -->
<p:fragment autoUpdate="true">
      <p:growl id="growl" showDetail="true" autoUpdate="true"
sticky="false" life="5000" for="flutuante"/>

</p:fragment>
</div>
```

Utilizando esse novo esquema de mensagens resolve o problema que temos com mensagem mostradas em processamentos ajax.

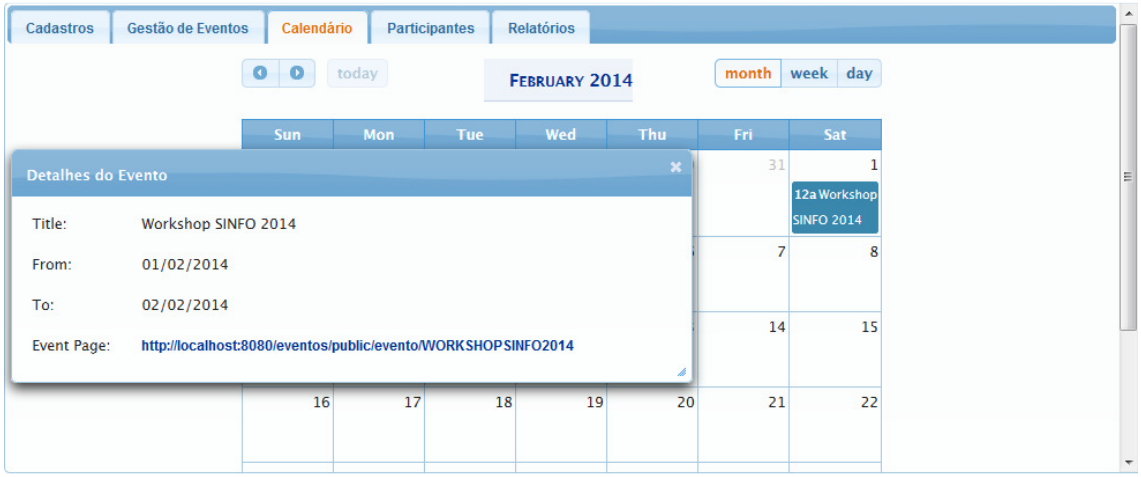
Hoje é necessário incluir a página erros_ajax.jsp e utilizar os métodos *addMensagem*Ajax()*, e ainda assim, as mensagens ficam duplicadas.

SigEventos 3.0.0

(com JSF 2.1, PrimeFace 4.0 e Prettyfaces 2.0*)*



Principal



Principal

Template padrão das páginas: Tudo 100% JSF, cabeçalho, rodapé, etc.

```
<!-- Include the application layout  -->
<h:body styleClass="background">


    <div id="container">

        <div id="cabecalho">

            <ui:include src="/WEB-INF/include/pages/cabecalhoInterno.xhtml" />

        </div>

        <div id="mensagem">

             <!--  Mensages in the top of the page -->
             <p:messages id="messages" showDetail="true" autoUpdate="true" closable="true" />

             <!--  Mensages the keep 5s in the same place when you roll the page (use when the page is very big, to user doesn't need page up to see the message ) -->
             <p:growl id="growl" showDetail="true" sticky="false" life="5000" rendered="#{requestScope.mostrarMensagenFlutuante}"/>

        </div>

        <div id="conteudo">
            <ui:insert name="conteudo">Conteúdo da Aplicação</ui:insert>
        </div>

        <div id="rodape">
            <ui:include src="/WEB-INF/include/pages/rodapeInterno.xhtml" />
        </div>

    </div>

</h:body>
```

login.xhtml usando o template padrão :

```
<?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:f="http://java.sun.com/jsf/core"
    xmlns:ui="http://java.sun.com/jsf/facelets"
    xmlns:a4j="http://richfaces.org/a4j"
    xmlns:rich="http://richfaces.org/rich"
    xmlns:p="http://primefaces.org/ui">

    <!--  User the cabecalho.xhtml template to define the login page -->
    <ui:composition template="WEB-INF/include/pages/templateInterno.xhtml">

       <ui:define name="conteudo">

              Conteúdo da Página de Login aqui !!!

       </ui:define>

    </ui:composition>


</html>
```

Principais vantagens encontradas com o JSF 2.X

Mais componentes da interface para melhorar a usabilidade do sistema.

Compatibilidade com os navegadores mais novos.