Displaying Errors - Infos - Warnings in JSF pages

{scrollbar}

One common question about JSF is how to manage messages of various types, which are generated by JSF components and backing beans. This discussion is designed as an introduction to the topic and focused on entry to mid-level JSF users.

The pattern described here is just one approach that we developed which meets our needs for:

1. Providing a consistent and generic means for including processing messages in all JSF pages.
2. Decorating the messages with images and styles to differentiate severity levels to the user which is browser safe.
3. Add additional messages when appropriate.
4. Be localizable.
5. Not affect page layouts when no errors are generated.

Discussion

Our user-interface requirements dictate that (a) as far as the user is concerned there is no such thing as an error, (b) the system should provide as little information as necessary to draw the user's attention to any issues raised during processing and help them correct the problem, (c) inform on all operations which change state in the back-end (success as well as failure), and (d) provide a way out (i.e. graceful means for backing out of any failed operation).

While (a) and (d) require detailed discussion on overall system design which is out of scope of this topic, (b) and (c) are certainly within the purview of this JSF discussion and are our points of focus.

JSF message components

The basic mechanism for managing page-specific messages is to include <h:message> and <h:messages> tags in the JSF page. <h:message> is component specific (i.e. tied to a particular JSF component ID) while <h:messages> can provide global messaging as well as roll-up all component specific messages. Our experience is that <h:message> is useful for components which have validation but the <h:messages> tag needs to be better managed to control layout and information-provision. For example, rather than displaying a list of validation errors that have been generated we would like to provide a new message such as 'One or more fields have missing or invalid values. Please re-check the fields indicated below and re-submit the form.'

JSF message tags display messages which have been queued as the request is processed. Each message that is queued has a message priority which is one of the FacesMessage priorities defined in javax.faces.application.FacesMessage and are defined as

java public static final FacesMessage.Severity SEVERITY_INFO = new Severity("Info", 1); public static final FacesMessage.Severity SEVERITY_WARN = new Severity("Warn", 2); public static final FacesMessage.Severity SEVERITY_ERROR = new Severity("Error", 3); public static final FacesMessage. Severity SEVERITY_FATAL = new Severity("Fatal", 4);

and can be either associated with a component or not. When a message is generated during processing, the component, validator or backing generating the message palces the message into one of the message queues where it can be accessed by the JSF during the rendering phase.

Generating Messages

Messages (error/info/warn/fatal) can be enqueued by calling the FacesContext.getCurrentInstance().addMessage() method. Our requirements state that messages are displayed on all change in state as well as processing errors such as bad databases retrievals etc. which means that the majority of our pages need to be able to generate messages. To this end, we found it is convienient to put quick-access methods in a base class and sub-class all backing beans from it. This is similar to how Sun Creator's 1.0 manages backing beans and our methods are adopted from this. For severity our base class has two methods as below for INFO.

java protected void setInfoMessage(String summary) { getFacesContext().addMessage(null, new FacesMessage(FacesMessage.SEVERITY_INFO, summary, null)); } protected void setInfoMessage(UIComponent component, String summary) { getFacesContext().addMessage(component.getClientId (getFacesContext()), new FacesMessage(FacesMessage.SEVERITY_INFO, summary, null)); }

The first method enqueues a global message with INFO severity while the second method associates the INFO message with a component (most probably from a custom (page-specific) validation. We also have a base class for all our validators (which perform page-specific validation when necessary) which uses the following protected method for raising messages

java protected static void invalidateInput(UIInput uii, String message) { uii.setValid(false); FacesMessage facesMessage = new FacesMessage (FacesMessage.SEVERITY_WARN, message, null); throw new ValidatorException(facesMessage); }

Note that we only raise messages with WARN severity as our requirements preclude displaying anything which looks or smells like an error to our users.

As all our backing beans have these methods accesible to them, we can simply add appropriate calls to enqueue messages in our business logic such as

java setInfoMessage(Messages.getString("MANIFEST_SAVED")); or setWarnMessage(Messages.getString("UPDATE_DATABASE_ERROR")); or setInfoMessage(Messages.getString("STORAGE_TRANSFER_ONE", new Integer(count), fromUnit.getName(), toUnit.getName()));

where the Messages class provides access to a messages properties file and performs standard parameter replacement using Apache's Common Lang StringUtils.

Displaying Messages

Rasing messages is one topic, displaying them another. As we need a consistent look and feel for all messages we choose to include a JSP in each page which handles all messages which are not component specific. Our PageMessages.jspf is as follows:

xml <%@ 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"%> <h:panelGrid styleClass="group" columns="2" cellpadding="2" cellspacing="0" width="100%" columnClasses="buttonCol, leftAlignCol" rowClasses="vertAlignTop" rendered="#{! empty facesContext.maximumSeverity}" > <h:graphicImage value="#{PageMessages.messageImage}" style="float: left; vertical-align: top;" /> <h:panelGrid columns="1" cellpadding="2" cellspacing="2" columnClasses="leftAlignCol" rowClasses="vertAlignTop" width="100%"> <h:outputText value="#{PageMessages.messageHeader}" escape="false" rendered="#{PageMessages.renderMessage}"/> <h:messages errorClass="errorMessage" infoClass="infoMessage" layout="table" globalOnly="true" showDetail="false" showSummary="true"/> </h:panelGrid> </h:panelGrid>

The messages are only displayed only if one or more messages are enqueued by conditionally rendering the table using `rendered="#{! empty facesContext.maximumSeverity}"` which uses EL to check if the queues are empty or not. Depending on the level of severity we display an themed image followed by an optional pre-amble to the messages and then the global messages themselves. The global messages are displayed in table format.

You will notice that we use a backing bean (PageMessages.java) in request scope to control the JSPF. The code for this is as follows:

PageMessages.java import Messages; import org.apache.commons.lang.StringUtils; import javax.faces.application.FacesMessage; import javax.faces.application.FacesMessage.Severity; public class PageMessages extends AbstractUIBean { private static final long serialVersionUID = -6479897299239746841L; private static final String BEAN_NAME = PageMessages.class.getName(); private String messageHeader; private String messageImage; private Severity severityLevel; public PageMessages() { messageHeader = null; // See if there are messages queued for the page severityLevel = getFacesContext().getMaximumSeverity(); if (null != severityLevel) { getLogger().debug("Severity Level Trapped: level = '" + severityLevel.toString() + "'"); if (severityLevel.equals(FacesMessage.SEVERITY_ERROR)) { messageHeader = Messages.getString("PAGE_MESSAGE_ERROR"); messageImage = "resources/warning.gif"; } else if (severityLevel.equals(FacesMessage.SEVERITY_INFO)) { messageHeader = null; messageImage = "resources/information.gif"; } else if (severityLevel.equals(FacesMessage.SEVERITY_WARN)) { messageHeader = null; messageImage = "resources/warning.gif"; } else if (severityLevel.equals(FacesMessage.SEVERITY_FATAL)) { messageHeader = Messages.getString("PAGE_FATAL_ERROR"); messageImage = "resources/stop.gif"; } } else { getLogger().debug("Severity Level Trapped: level = 'null'"); } } public Boolean getRenderMessage() { return new Boolean(StringUtils.isNotBlank(getMessageHeader())); } public String getBeanName() { return BEAN_NAME; } public String getMessageHeader() { return messageHeader; } public String getMessageImage() { return messageImage; } }

Our backing bean extends our basic UI base classes which provides access to central logging etc. You can see that if a component has enqueued a message of ERROR we add a localized message at the top of the page which (we hope) gracefully moderates the user-impact of seeing the same form again with a bunch of annotations!

The PageMessages.jspf is included into each JSP using the following code:

xml <f:subview id="messages"> <jsp:directive.include file="PageMessages.jspf" /> </f:subview> {scrollbar}