How the immediate attribute works

{scrollbar}

Purpose

The immediate attribute can be used to achieve the following effects:

- Allow a commandLink or commandButton to navigate the user to another page without processing any data currently in input fields of the current screen. In particular, this allows navigation to occur even when there are currently validation errors. A "cancel" button typically falls into this category.
- Allow a commandLink or commandButton to trigger back-end logic while ignoring validation for some of the fields on the screen. This is a more general version of the item above.
- Make one or more input components "high priority" for validation, so that if any of these are invalid then validation is not performed for any "low-priority" input components in the same page. This can reduce the number of error messages shown.

Overview

Before we can discuss immediate, we first need to review the JSF request processing lifecycle.

- **Restore View** - Creates or restores the previous page.
- **Apply Request Values** - Set component submitted values to request values.
- **Process Validations** - Convert and validate component values. Set component values to submitted values if valid.
- **Update Model Values** - Set backing bean values to component values.
- **Invoke Application** - Execute actionListeners and actions.
- **Render Response** - Return response.

Many developers are under the impression that using the immediate flag on a component skips the "Process Validations" phase. It doesn't. What using immediate does is cause a component to be processed in the Apply Request Values phase.

UIInput Components

Using immediate means the component value will be validated during apply-request-values, ie before any non-immediate component values (which validate in process-validations). Any validation error in the set of input components that are marked immediate will cause processing to move to the render phase after the apply-request-values phase is complete, meaning that if any immediate component fails validation then errors associated with non-immediate components will not be shown. In addition, if the new value of an immediate input component is different from the existing value specified by the value attribute then a ValueChangeEvent is raised (as normal), but this event is processed at the end of the !ApplyRequestValues phase, NOT at the end of the ProcessValidations phase. In particular, this means that any ValueChangeListener associated with the component will execute before any immediate UICommand component's ActionListener (assuming the command component occurs later in the page).

Marking an input component as immediate does NOT affect the model update; any new data is still pushed into the model at the Update Model phase (ie after any immediate command components have executed). Note, however, that the !ValueChangeListener could be used to update the model directly.

UICommand Components

Using immediate causes the component ActionListener or action-method to be executed at the end of the apply-request-values phase, ie before any non-immediate value validation and before any backing bean updates (update-model phase).

If the action method is of a form that returns a navigation string, then:

- Any non-null string will cause the lifecycle to proceed directly to the render-response phase, meaning that validation of any non-immediate input components never occurs. This is why an immediate command component is a natural way to implement "cancel" operations; it works even when there are input fields in the page that would fail validation. Of course as a result, there is never any update-model phase, ie data entered by the user is discarded.
- A null return value causes processing to continue as normal, ie non-immediate components are validated then update-model is executed (if no validation errors occurred).

For an action listener method that returns void, it is necessary to call

java facesContext.renderResponse();

if the normal flow is not desired.

The most significant issue when using an immediate input component is that new input data entered by the user is not usually available from the model as the update-model phase has not yet executed.

For non-immediate input components in the page, the only way an action method for an immediate command component can access user input data is by using component-binding or lookup-by-name to retrieve a specific UIComponent object then calling getSubmittedValue() to obtain the raw string provided by the user. This value has not been converted to its target type (using a user-specified or default Converter), nor has it been validated.

For immediate input components, the conversion and validation steps have been executed; using the corresponding UIComponent it is possible to get the converted value. Alternatively, if the component is "before" the UICommand component in the page, and has a !ValueChangeListener attached then that will have executed so any side-effects of that method (including direct update of a backing bean) can be relied upon to be available. Note, however, that if the component fails validation then no backing bean update will have been performed.

Warning: if the action method updates the model but does not perform navigation then any change to a backing bean value will be overwritten when the input component value goes through validation and update model.

Any immediate input components on the page that fail their validation will NOT stop immediate command components from executing; this is quite different from the behaviour for non-immediate input and command components.

See Also

- Clear Input Components

Limitations

Using immediate for anything other than the trivial case of a cancel button is problematic.

Solution 1

First mark the relevant command component as immediate, then for each input component that the associated action method needs to implement:

- mark the input component as immediate
- add a valueChangeListener attribute

xml <h:inputText value="#{pageBean.foo}" immediate="true" valueChangeListener="#{pageBean.setFoo}"/>

The referenced method looks like this:

java // normal property setter public void setFoo(String val) {...} // immediate input hack: update model at apply-request, not update-model public void setFoo(ValueChangeEvent ev) { setFoo((String) ev.getNewValue()); // prevent setter being called again during update-model phase ((UIInput) ev. getComponent()).setLocalValueSet(false); }

This effectively moves the update-model behaviour for the modified input component from the update-model phase into the apply-request-values phase. An immediate button can then access these values just like a non-immediate button accesses its inputs - except that the action listener method does need to manually check whether the validation failed for this component (in which case the ValueChangeListener never runs).

In the case where the input component's validation does fail, and the (immediate) action listener does not force navigation then the page will be redisplayed with only the immediate validation errors (not errors from any non-immediate components); this may be exactly what is wanted.

Solution 2

Have the action method invoked by the component access the raw components. However:

- You will not have access to converted or validated component values on non-immediate components.
- You will not have access to updated backing beans. You cannot change the values for non-immediate components (except by changing the raw submitted values).
- The Update model phase may not be executed.
- Backing beans that non-immediate components reference might or might not be updated.

Solution 3

Don't make the command immediate at all; instead make the validation check whether it should run or not (see also the link below to the Optional Validation Framework).

For example, if the problem is a required field, then do something like this (untested code!):

xml <h:inputText required="#{!pageBean.cancelling}"/> java public boolean isCancelling() { // assumes cancelButton is a component binding FacesContext fc = FacesContext.getCurrentInstance(); Map reqParams = fc.getExternalContext().getRequestParameterMap(); return reqParams.containsKey (cancelButton.getClientId()); }

Solution 4

Use the tomahawk subform component. (The OptionalValidationFramework solution is no longer recommended or supported).

Other Relevant Information

Non-validated EditableValueHolder (UIInput) components render their submitted value, not the backing bean or local value.

Since only one group of components can be marked as immediate, you cannot have multiple kinds of immediate actions that depend on different component sets.

{scrollbar}