Learn hidden but valuable lessons about extending JSF code.

Mastering OmniFaces

A Problem to Solution Approach

Reviewed by
Bauke Scholtz & Arjan Tijms

To make JSF life easier
Omnify your JSF applications

Take JSF to the next level
bug fix, gaps, pitfalls, JSF design patterns

Constantin Alin **Anghel Leonard**

THE HIDDEN TABLE OF CONTENT

5 October 2015

Chapter 1 - OmniFaces Components

1.1 OutputLabel

- registering a custom component for listening `PostRestoreStateEvent`
- programmatically extracting the value of an attribute
- searching components using relative-up/down search algorithm
- collect the value of the `label` attribute for the given UI component
- setting an attribute value as a literal or as a value expression

1.2 Param

- instructing an `UIComponent` to support a JSF converter
- sample of extending the `ValueHolder` interface (`ParamHolder`)
- having a quick overview of how to use `StateHelper` class
- introducing static vs. dynamic converters
- temporarily replacing the default output response (`ResponseWriter`)

1.3 OutputFormat

- working with attributes of type `var`
- suppress the invocation of `setValueExpression()` for an attribute
- replacing/restoring the `ResponseWriter` in the rendering process
- carrying an artifact via the `FacesContext` attributes map
- putting an artifact in the request map (scope)

1.4 Messages

- programmatically indicating the renderer of a component
- indicating that a component will render its children also
- understanding the triplet: `encodeBegin()`, `encodeChildren()`, `encodeEnd()`
- programmatically collecting the available messages – `FacesMessages`
- discovering the OmniFaces utility, `findComponentsInChildren()`
- storing/removing something in/from request scope (request map)

- replacing/restoring something from/in request scope (request map)
- "postponing"/triggering the rendering of a component
- programmatically generating the markup of a HTML table/list

1.5 OnloadScript

- working with the `@ListenerFor` annotation
- rendering the `<script>` element
- determining if the current request is/isn't an AJAX request
- determining if the value of the `render` attribute is `@all`
- programmatically subscribe to view events (e.g. `PreRenderViewEvent`)
- using the `OmniPartialViewContext` for executing the given scripts on
- completion of the current AJAX response

1.6 DeferredScript

- marking a resource as rendered
- programmatically creating a `Resource`
- rendering the `<script>` element when a resource exist
- rendering the `<script>` element when a resource does not exist

1.7 Highlight

- working with JSF `VisitTree` API
- access, from JavaScript, the HTML markup via the *clientIds* provided by JSF

1.8 ViewParam

- transforming a stateful component into a stateless component
- suppressing decoding and validation at postback requests
- suppressing the built-in `required` validator
- using the `name` attribute when the `label` attribute is absent
- working with the OmniFaces utility, `MapWrapper`

1.9 Form and IgnoreValidationFailed

- working with several OmniFaces utility methods
- writing a custom `FacesContext`, `Application` and `ViewHandler`
- working with `ViewHandler#getActionURL()`
- encoding URLs
- programmatically control Process Validations phase (via `processValidators()`) and Update Model Values phase (via `processUpdates()`)
- subscribing with a `Callback` to after Restore View phase
- a good example for understanding the JSF lifecycle

1.10 Cache

- JSF post-processing tips
- subscribing with `Callback`s to view/phase events
- working with `BufferedHttpServletResponse`
- introducing `TagHandler`s
- working with `VariableMapper`
- overriding `UIComponent#isVisitable()`

1.11 CommandScript

- programmatically loading the JSF AJAX library
- writing a custom `UICommand`
- extracting all `UIParameter`s nested in a certain component
- validating if a component has a parent of a certain type
- resolve the given space separated collection of relative *clientIds* to absolute *clientIds*
- creating and queuing an `ActionEvent`
- adding the `autorun` feature

1.12 ResolveComponent

- introducing "facelet scope"
- passing `FaceletContext` to a custom `UIComponent`
- storing a `ValueExpression` in the "facelet scope"

- storing a `UIComponent` in request scope
- subscribing to `PreRenderViewEvent` and `PostRestoreStateEvent`
- writing a custom `ValueExpression`
- working with `ReadOnlyValueExpression`
- writing a functional `Callback`
- finding a component in the component tree by *clientId*

1.13 ResourceInclude

- working with OmniFaces, `validateHasNoChildren()` utility method
- obtaining the environment-specific object instance for the current request/response
- building a buffered response via OmniFaces, `BufferedHttpServletResponse` using the Servlets API `RequestDispatcher` to includes the content of a resource

1.14 GraphicImage

- understanding how JSF processes images
- working with JSF resource handlers
- writing a cacheable dynamic resource
- working with images request path
- understanding how "last modified" works
- obtaining/determining image content type
- obtaining the data URI from an image
- parsing/evaluating a `ValueExpression` and store
- converting a `ValueExpression` into a request path eligible later for Java Reflection API
- parsing a request path and invoking a method using Java Reflection API
- testing the presence of an annotation on a class
- converting the given strings to objects using converters registered on given types
- converting the given objects to strings using converters registered on given types

1.15 ComponentIdParam

- extending `ViewParam`
- processing request query parameters

- understanding `decode()` method goal
- suppressing validations and data model updates
- writing a custom `PhaseListener`
- learning about quick succession of writings
- turn off/on the original response writer

1.16 MoveComponent

- moving components in the component tree
- moving facets and behaviors
- understanding how behaviors works in JSF
- implementing `ClientBehaviorHolder`
- subscribing to `PreRenderViewEvent`, and `PostAddToViewEvent`

1.17 Tree

- exploring several `Tree` use cases
- controlling the `processDecodes()`, `processValidators()`, `processUpdates()` triggers and the `encodeChildren()` from a single place (`TreeFamily`)
- using a "bunch" of OmniFaces utilities which are very handy in day by day development
- following the JSF flow in a recursive implementation
- working with the "special" attributes, `var` and `varNode`
- exploring how the events are queued and broadcasted

1.18 ConditionalComment

- introducing `javax.faces.FACELETS_SKIP_COMMENTS`
- understanding the `escape` attribute role
- defining a Java reserved word as an `enum` item

Chapter 2 - OmniFaces Validators

2.1 JsfLabelMessageInterpolator

- introducing Bean Validation
- obtaining the default message interpolator
- customizing the default message interpolator

2.2 ValueChangeValidator and ValueChangeConvertor

- writing a template validator
- writing a template converter
- check if a component supports validators/converters (is an instance of `EditableValueHolder`)
- programmatically obtain the model value of a component
- explicitly invoke the converter `getAsString()` method

2.3 RequiredCheckbox Validator

- implement a custom validator
- programmatically check a component type
- programmatically locate a message bundle and access its keys

2.4 Validate Bean

- implement a validator as a `TagHander`
- good lesson for understanding and manipulating JSF lifecycle phases
- what's a JSF new component in the component tree
- get the value of an attribute as a `ValueExpression` to be carried around and evaluated at a later moment in the lifecycle without needing the Facelet context
- postpone a task before/after a certain JSF lifecycle phase of the current request
- get the closest parent of the given parent type

- write a `SystemEventListener` for `PreValidateEvent` and `PostValidateEvent` fired by `UIInput`s that have attached a `BeanValidator`
- finding a certain validator in the list of validators of a component
- use component attributes mutable `Map` to pass objects between methods
- programmatically indicate/replace/restore the `BeanValidator` validation groups
- use OmniFaces `Copier` API for copying objects
- traverse a form children and extract the `ValueReference` of each `EditableValueHolder` for the specified base
- use `javax.validation.Validator` in JSF

2.5 ValidateUniqueColumn

- implementing a custom validator as a `TagHandler` that is registered as a listener for `ValueChangeEvent`
- what's a JSF "new" component in the component tree
- `ValueChangeEvent` internal overview
- JSF `VisitTree` implementation sample - visit a JSF table rows (`UIData` iteration)
- tips for collecting the values of an `UIInput` component in an `UIData` component
- signaling invalid values

2.6 Validate*Xxx*

- writing a validator as a custom component
- "intercepting" Apply Request Values phase via `processDecodes()` method
- "intercepting" Process Validators phase via `processValidators()` method
- "intercepting" Update Model phase via `processUpdates()` method
- collecting inputs (and their values) from a form
- marking invalidation at component/context level
- associating the message bundle keys with custom components `COMPONENT_TYPE`s
- programmatically accessing an annotation (e.g. `@FacesComponent`)
- programmatically accessing message bundle
- writing a component handler, and access `ValueExpression`s/`MethodExpression`s representing the values of attributes of the component

- "ugly" , but functional, approach for distinguishing between a `ValueExpression` and a `MethodExpression`
- evaluating a `ValueExpression`
- invoking the method indicated via `MethodExpression`

Chapter 3 - OmniFaces Tag Handlers

3.1 Convertor/Validator

- introducing JSF converters/validators handlers
- exposing the `TagHandler` protected methods as public methods
- using `binding` and `converterId`/`validatorId` attributes to instantiate a certain converter/validator
- collecting values of attributes as `ValueExpression`s
- using the `java.beans.Introspector` API to identify the setters of a bean
- wiring attributes values with corresponding setters (literal text and deferred value expressions)
- using Java Reflection API to invoke setters
- storing deferred value expressions and setters in a `Map`
- creating and using an anonymous `Converter`/`Validator`

3.2 ImportConstants/ImportFunctions

- exploiting several features of Java Reflection API
- creating and working with a cache based on the `java.util.concurrent.ConcurrentHashMap`
- storing entries in request scope
- working with Facelet scope
- collecting the values of the `var` and `type` attributes

3.3 ViewParamValidationFailed

- writing a validation handler via a tag handler "local" (via `UIInput#isValid()`) and "global" (via `FacesContext#validationFailed()`) validation
- programmatically redirecting and sending HTTP status error code
- subscribing the given `Callback` instance to the given component that get invoked only in the current request when the given component system event type is published on the given component
- collecting messages from faces messages list
- add a message in flash scope
- avoid the Faces message has been enqueued but is not displayed warning
- evaluating the given value expression as a string

3.4 EnableRestorableView

- understanding the `ViewExpiredException`
- understanding the view restoring process
- writing a custom view handler
- creating a new view via `ViewHandler#createView()`
- building a view via `ViewDeclarationLanguage#buildView()`
- writing a custom `FacesContext`
- programmatically attaching/detaching a custom `FacesContext`
- obtaining the view associated render kit

3.5 MassAttribute

- checking at runtime the required attributes via `TagHandler#getRequiredAttribute()`
- working with `TagHandler#nextHandler` field
- programmatically setting an attribute of a component

3.6 MethodParam

- great lesson about EL API
- wrapping a `ValueExpression` into a `MethodExpression`
- developing a custom `ELContext`

- developing a custom `ELResolver`
- working with `VariableMapper`

3.7 TagAttribute

- working with Facelets files
- introducing Facelet context (e.g. `DefaultFaceletContext`)
- introducing VariableMapper (e.g. `VariableMapperWrapper`)
- understanding how Facelets files are processed by JSF
- writing and setting a custom `VariableMapper` (`DelegatingVariableMapper`)

Chapter 4 - OmniFaces Converters

4.1 GenericEnumConvertor

- writing a custom `EnumConverter`
- capturing the `enum` type in `getAsString()`
- storing/accessing the `enum` type in/from view map

4.2 SelectItemsConvertor and SelectItemsIndexConvertor

- introducing `SelectItem`, `SelectItemGroup`, `UISelectItem` and `UISelectItems` API
- extracting all `SelectItem` expressed via `UISelectItem` and `UISelectItems`
- learning the algorithm expressed by a `UISelectItems` component that uses the `var` iterator construct to generate a list of `SelectItems`
- working with OmniFaces, `ScopedRunner`
- iterating `SelectItemGroup`
- working with `FacesContext` attributes

Chapter 5 - OmniFaces Exception Handlers

5.1. FullAjaxExceptionHandler

- extracting unhandled exceptions
- handling `AbortProcessingException`
- locating error pages in `web.xml`
- programmatically creating and rendering a new view

5.2 FacesMessageExceptionHandler

- accessing unhandled exceptions
- turning each exception into a global FATAL faces message

Chapter 6 - OmniFaces Exception Contexts

6.1 OmniPartialViewContext

- extending the `PartialViewContext`
- "dissecting" the partial rendering response
- understanding how a partial response is rendered via `PartialResponseWriter`
- writing a custom `PartialResponseWriter`
- altering the partial response content
- using and understanding `<eval>` tag
- understanding how Mojarra and Apache MyFaces are working with partial response
- programmatically inspecting the `web.xml` file
- check if the `web.xml` security constraint has been triggered during this AJAX request
- resetting and closing the AJAX response

Chapter 7 - OmniFaces Resource Handlers

7.1 CDNResourceHandler

- fortify your knowledge about the JSF `ResourceHandler` and `Resource` APIs
- working with the OmniFaces `DefaultResourceHandler` API
- collecting the context parameters from `web.xml`/`web-fragment.xml`
- replacing a resource request path
- using the `Application#evaluateExpressionGet()`
- introducing the OmniFaces `RemappedResource`

7.2 UnmappedResourceHandler

- replacing the mapping prefix/suffix of the original generated request path
- understanding the `isResourceRequest()` and `handleResourceRequest()`
- recognizing PrimeFaces dynamic resource requests

7.3 CombinedResourceHandler

- collecting component resources by their type
- hacking RichFaces 4 component resources
- combining multiple component resources in a single one
- using server-side cache for the combined component resources
- determining the render type for a component resource
- sequentially serving multiple resources
- modifying an existing component resource
- creating a new component resource

Chapter 8 - OmniFaces Event Listeners

8.1 InvokeActionEventListener

- understanding how `<f:event>` works
- adding support for new `<f:event>` types (`preInvokeAction` and `postInvokeAction`)
- writing a custom phase listener
- subscribing a phase listener to other events (e.g. `PostValidateEvent`)
- collecting components that have certain registered listener
- introducing the OmniFaces, `DefaultPhaseListener`

8.2 ResetInputAjaxActionListener

- writing an artifact that can act as an action listener or as a phase listener
- using `processAction(ActionEvent)` method
- getting partial executing/rendering IDs
- introducing built-in, `resetValue()`
- chaining action listeners invocation
- seeing another useful Visit Tree API implementation

Chapter 9 - OmniFaces View Handlers

9.1 NoAutoGeneratedIdViewHandler

- introducing JSF auto-generated IDs
- writing a custom view handler
- checking the project stage
- writing a custom `ResponseWriter`
- overriding `ResponseWriter#cloneWithWriter()`

- distinguish between "normal" IDs and "special" IDs (e.g. view state and client window IDs)

Chapter 10 - There's More

- brief overview of other OmniFaces artifacts
- what's new in OmniFaces 3.0