International Journal of Computer Trends and Technology
ISSN: 2231-2803 / https://doi.org/10.14445/2231280453/IJCTT-V67I3P123
Volume 67 Issue 3,119-123, March 2019

Original Article

Apache Struts 1.x based Enterprise System Transformation to JEE 7+, JSF and Prime Faces based Components

Vijay Kumar Pandey

Director of Technology Solutions, Intueor Consulting, Inc. Irvine, CA (United States of America)

Abstract *- The paperis intended to provide an elaborate understanding of how to transform open source Apache Struts 1.x (has reached End of Life – EOL) based enterprise system to JEE7+ and PrimeFaces based components such as Java Server Faces, Bean Validation and Context Dependency Injection (CDI). Apache Struts 1.x during early 2000 was the defacto standard to build web application using Java Server Pages (JSP), Struts tags, Tiles, Struts Action classes, Struts based custom tags, Struts based validations, any struts xdoclet tags for configuration along with other Struts dependent components.*

Keywords *- Apache Struts, Java Platform Enterprise Edition 7 (JEE7), Java Server Faces (JSF), Facelet, XHTML, Java Server Pages (JSP), Bean Validation, CDI, Tiles, XDoclet, Action, PrimeFaces (PF), OmniFaces, Expression Language (EL)*

I. INTRODUCTION

One of the most important aspect of the transformation is understanding of the major difference between the Struts and JSF based web components life cycle. Apache Struts is a Request-Response based Framework while JSF is a Component-based Framework.

II. WEB PRESENTATION LAYER COMPONENTS

The presentation layer components in Struts based enterprise system will mainly consist of JSP, Tiles configuration, Struts based tag library, Struts based custom tag library along with any custom javascript and cascading style sheet (css). These Struts components should be transformed to following JEE 7/JSF based components.

- *JSP* – XHTML page based on Facelet
- *Tiles Configuration* - No corresponding tiles configuration in JSF. Various tiles related attributes can be part of the xhtml itself. By using "template" attribute of "ui:composition" along with "ui:param" attributes, tiles attributecan be transformed except for the navigation, which will be discussed in the Navigation section.
- *Javascript* -JavaScript's (jQuery + usage of PF widgets)
- *Struts Tags* - Usage of Facelet tag libraries along with the usage of PF component library and OmniFaces JSF utility library.
- *Custom Struts based Tags* - These should be replaced with new custom JSF components or JSF based tag library.

III. REQUEST CONTROLLER COMPONENTS

The request controller layer in Struts ismainly comprised of Struts action, action form, form validations, xdoclet tags (mainly used for struts and validation config). These components are converted to various JSF and JEE7 based components. The following outlines the nature of the conversion.

- *Action Classes* – JSF based and CDI annotated Controller classes.
- *Action Form Classes* - JSF based and CDI annotated Controller bean classes.
- *XDoclet Tags* – Replace with relevant annotations on controller classes. Validation xdoclet tags to be replaced with appropriate Bean Validation constraint annotations.

A. Action to Controller Class Transformation

This section outlines various transformation needed for action classes and their associated config, which could have been defined by xdoclet tags or were manually defined in struts config.

- *Action Path* – JSF based Controller classes should be CDI enabled and can be annotated with CDI based annotations such as Named, RequestScoped, ViewScoped, SessionScoped.
- *Any custom config property* – Create a custom annotation to manage these custom properties
- ActionForward - in JSF, there is no concept of forward object (ActionForward in struts); simply return the actual forward value (think of tiles def name in struts), which is termed as an outcome.
- *LookupDispatchAction:getKeyMethodMap* – This is not needed as facelet command buttons can

directly invoke the controller method through EL expression.

- *ActionForm references in Action* - In Struts Action classes, *ActionForm* are created by the Struts framework and passed as a parameter to the *execute* or other custom method (executed through *LookupDispatchAction*). In JSF,*ControllerBeans* are either created through a CDI *PostConstruct* annotated method, or they are injected in the controller, where they are either *RequestScoped* or *SessionScoped*. These controller beans are instance variables of Controller objects.
- *Action Class Methods*–Action classes have *execute* and other custom methods related to *LookupDispatchAction*, and they will have method arguments as *ActionMappingActionForm*,*HttpServletRequest*,*HttpServletResponse*.Thesemethod signatures must be transformed by removing all the method arguments. Any scenario, which needs access to request and response, can be managed through JSF's *FacesContext* object (at a high level, it is a wrapper for request and response objects). Any other nonstandard methods that used Struts related arguments must be transformed in a similar manner.
- *ActionForward (return type)* - The return is now a String type value for outcome (like 'tiles def name') or it can be void, if no navigation, and then return is to the same view.

B. Action Form to Controller Bean Class Transformation

This section outlines various transformation needed for action form classes, validate method and associated validation config.

- *Action Form* – Struts *ActionForm* can be converted to *ControllerBean* classes. These classes are available as instance variables in Controller classes, but in some scenarios these classes are CDI beans, then they are injected in controller classes. CDI bean scopes are:

(1) RequestScoped
(2) SessionScoped (cases where struts action config scope is session)

- *Action Form Validation Config (with or without XDoclet)* - These can be transformed to use JavaBeans constraint annotations at the field level.
- *Validate method* - This method signature must be changed. Both the ActionMapping and HttpServletRequest needto be removed and replaced with FacesContext.

C. Struts Message Resources & Internationalization Transformation

JSF provides a mechanism to provide the resource bundle feature as part of the *faces* configuration and individually at a page level, but then bean validation will not be able to hook into the JSF provided resource bundle. To overcome this limitation, resource bundles can be set up through spring framework, then resource bundle can be easily retrieved through EL expression in facelets.

Excerpts from spring context xml for message resources set up

```
<bean id="messageLocator"
class="org.springframework.validation.beanvalidation.MessageSourceResourceBundleLocator">
<constructor-arg index="0" ref="messageSource" />
</bean>

<bean id="messageSource" class =
"org.springframework.context.support.ReloadableResourceBundleMessageSource " >
<property name="useCodeAsDefaultMessage" value="true" />

<property name="alwaysUseMessageFormat" value="true" />
<property name="basenames" value="#{ propHolder.basenm}" />
</bean>

<bean id="propHolder" class =
"some_custom_class_with_array" >
<property name="basenm">
        <list>
        <value>classpath:msgRes1</value>
        <value>classpath:msgRes2</value>…
        </list>
        </property>
<bean>
```

D. Struts Locale& Internationalization Transformation

JSF handling of locale is based on the view (think facelet and xhtml).Since message resources are set up through *Spring* (in the above step), the locale from the JSF must be propagated to *Spring* through *LocaleContextHolder*for *Spring* to pick the correct locale sensitive resource bundles.

E. Struts Navigation (Tiles, ActionForward) Transformation

JSF provides a mechanism of navigation rules and navigation cases to navigate to different views, while struts use a combination of tiles config, action forward or another struts config to determine the navigation. JSF navigation config file can be created which will be set up through web.xml using a context param *javax.faces.CONFIG_FILES*. Since each view can have its own set of outcomes (forwards in struts), this could easily lead to a large file size; to overcome this issue, all the tile defs can be created as navigation cases under a global navigation rule (from view id as *), so any view can refer to these outcomes globally. Specific navigation rules for certain views can be created on exception basis.

Sample Navigate Rule for Global access maps to tiles-def

```
<navigation-rule>
<from-view-id>*</from-view-id>
<navigation-case>
<from-outcome>.tiles_def_name</from-outcome>
<to-view-id>/faces/xyz/tiles_def_map.xhtml</to-view-id>
</navigation-case>….
</from-view-id>
```

In the above case, if the controller class method returns the outcome named "*.tiles_def_name*", JSF will navigate to *tiles_def_map.xhtml* for rendering

F. Struts RequestProcessorTransformation

If Struts *RequestProcessor* class was extended to provide custom navigation behaviour, then JSF based *ConfigurableNavigationHandler* can be extended to provide additional processing logic. This additional logic needs to go in.

```
public void handleNavigation(FacesContext context, String
fromAction, String outcome);
```

- *Navigation to other Action* – JSF's view rendering technology is Facelets (xhtmls), JSF will not directly navigate to other controller, it needs to navigate to the view (xhtml). To handle this scenario, a dynamic mechanism can be created.Create a dynamic xhtml, let's say*/faces/ /dynamic.xhtml*.This view needs to have a capability to take controller and its method name as an input to execute the controller method and then let the JSF navigate to the view based on the method execution outcome. Security check can be added to only allow execution of secure and authorized methods.

G. Component ID Generation in Facelet (Xhtml)

Struts html tags do not generate id attributes for their html controls until styleId or errorStyleId attributes were specified at the tag. It generates the "name" attribute for their html component based on the "name" and "property" attributes. JSF works on the "id" attribute and not "name".It does generate the "name" attribute on the rendered html component and most of the time its same as id.However, there are cases where they differ,such aswith radio button groups (same "name" attribute will define that group but will be still having different "id" – "id" cannot be same in an html document). JSF also has certain restrictions on what letters and characters can constitute an id. The section below will discuss certain scenarios for both Struts and JSF.

a) Simple Text Component

Struts: <html:text property="xyz" />

Struts will use the property attribute and create the "name" attribute for the html component as "xyz"

JSF (PrimeFaces based text component):
<p:inputText value="#{controllerBean.xyz}" id="xyz"/>

In JSF, if an id attribute is not provided, one will be auto generated. If there is no need to access this component by a specific known id, id attribute can be omitted. *{controllerBean.xyz}* is an EL expression for this component to retrieve and set values from the controller bean. This component will also generate the "name" attribute based on the "id", it will have the same id and name attribute generated for the html component which is "xyz" (NamingContainer is another JSF feature that will change the generated id based on the parent). It is important to note that when an html form is submitted, request parameters are formed based on the "name" attribute (whether its Struts or JSF). As part of the transformation process, "id" attribute can be generated if access is needed by a specific id, but the primary reason for generating them can be because of specific request parameter access by its name (which is based on id in JSF).

- ClientId - What is referred to as "id" value in the browser for the html component, is referred to as "clientId" in the JSF system. Since the "id" attribute value at the component level is not what is always generated, if the parent is the NamingContainer, these id's can change. Most of the time theprimary parent NamingContainer is the html form, if its prependId attribute is set to false; it will not affect the generated "id" and "name" attributes of the contained components.
- ID Naming Restriction - JSF specification sets certain restrictions on the naming of the id and they are:
  (1) It should not be an empty value
  (2) The first character must be a letter or underscore '_'
  (3) Second character onwards could only be a letter, digit, underscore '_' or dash '-'

b) Indexed Text Component

Usage of Struts html text tag inside a *c:ForEach* with indexed attribute set to true.

Struts:

```
<c:forEach items="${someList}" var="varBean"
varStatus="vatStatus">
    <html:text name="varBean" property="xyz" indexed="true" />
</c:forEach>
```

The above struts html loop will generate html text controls with name as *varBean[0].xyz*. The part of the name '[0]' will be a running sequence number based on the iteration index starting from zero.

The above JSP code can be transformed in two ways depending upon the requirement.

Scenario 1:

```
<ui:repeat value="#{someList}" var="varBean"
varStatus="vatStatus"><p:inputText value="#{varBean.xyz}"
id="varBean__xyz" /></ui:repeat>
```

In this case, there is no "id" attribute at *ui:repeat* level.To make the *xyz* related id's unique for the view, they are suffixed with the var attribute value of

ui:repeat and then separated by the "__", a specific value that can be used during transformation to segregate the naming container from the property id.*ui:repeat* being a NamingContainer, its auto generated id will prefixed in its child element, so the id and name generated for this will be of the form j_id_56:0:varBean__xyz. Here 'j_id_56' could be anything (normally j_id_sequence) and is the auto generated id of the ui:repeat. ':' is the separator character for naming container.Default is ':', which could be overridden through a servlet context param '*javax.faces.SEPARATOR_CHAR*'. '0' is the running iteration index of the ui:repeat value.If there was a need to update same name instance variable into the controller bean of the next view from the request parameters of current view, then id attribute needs to be specified at the *ui:repeat*, refer scenario 2 below for sample code.

Scenario 2:

```
<ui:repeat value="#{someList}" var="varBean"
varStatus="vatStatus" id="someList"><p:inputText
value="#{varBean.xyz}" id="varBean__xyz" /></ui:repeat>
```

In this case the id and name attribute for the *xyz* related html text controls will be generated as someList:0:*xyz*.

H. Faces Configuration

Faces configuration (faces-config.xml) is the main JSF config file that lets the JSF system configure various features for the given system. This is set up in web.xml under the servlet context param *javax.faces.CONFIG_FILES*. Configuration can be divided in multiple files, the value of this context param can be comma separated file names. If the file name is faces-config.xml, then it does not need to be stated in the web.xml. Some of the important elements that can be configured in this are System Event Listener, Navigation Handler, EL Resolver, Resource Handler, LifeCycle Phase Listeners, Behaviors, Converters, Managed Bean, Partial View Context Factory, Exception Handler Factory, Overridden Components, and Overridden Renderers.

I. WebXML Configuration

In addition to various listeners and filters that are configured in web.xml, several new configs should be added mainly the faces servlet, various servlet context params, error handlers etc.

Faces Servlet is added to handle requests related with xhtml and unhandled resources

```
<servlet>
    <servlet-name>facesServlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>facesServlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
    <url-pattern>/javax.faces.resource/*</url-pattern>
</servlet-mapping>
```

Some of the important web xml context params that could be added when moving away from Struts to JSF are

- primefaces.THEME – Theme used by PF
- javax.faces.DATETIMECONVERTER_DEFAULT_TIMEZONE_IS_SYSTEM_TIMEZONE - Use system timezone for data time related to faces converters
- javax.faces.SERIALIZE_SERVER_STATE - Indicates whether the view state should be serialized before saving in the session or not
- javax.faces.FACELETS_BUFFER_SIZE - The default buffer size for the servlet response
- javax.faces.FACELETS_REFRESH_PERIOD - Period used to refresh the actual facelet tree from the physical xhtml file
- javax.faces.PROJECT_STAGE – Production or Development
- javax.faces.STATE_SAVING_METHOD - Indicates the saving of the JSF state on the server
- javax.faces.PARTIAL_STATE_SAVING - Indicates if the partial state saving is turned on or not
- javax.faces.CONFIG_FILES - Comma separated faces configuration files.
- javax.faces.FACELETS_LIBRARIES - Indicates the various comma separated facelet libraries

IV. CONTEXT DEPENDENCY INJECTION (CDI)

In CDI 1.1 (JEE 7), it is not mandatory to provide beans.xml (in the war project, it is in WEB-INF); CDI is enabled by default. However, by providing beans.xml, there are some configurations that can be explicitly achieved that are not possible in an automated fashion.

A. Interceptors

With CDI 1.1, interceptors are enabled with priority, but having them declared in beans.xml provides a quick view of the interceptors and they can also be disabled by simple removing or commenting that entry. Some of interceptors that can be created could be for cross cutting concerns such as security. Controller and their methods could be annotated with security related annotations, which are backed by CDI interceptors to ensure only authorized users can execute that controller method. Interceptors can also come handy when there is a need to execute any generic method before the actual controller method execution.

B. Alternatives

Alternatives feature canbe used for those struts action classes transformation that have the same action path in different projects but built on the same base enterprise framework with different action configurations. To provide this type of different configuration but same CDI name, *Alternative* CDI feature can be used. All the *Alternatives* will still have the same CDI name, but which one takes precedence depends upon the entry in the beans.xml.

```
<alternatives>
   <class>
         <<CDI class name specific for the project>>
   </class>
</alternatives>
```

V. PRIMEFACES WIDGETS

Widget is a terminology that is used in PrimeFaces, comparable to JavaScript objects to their counterpart PF JSF components

A. Widget Naming

PF provides an attribute '*widgetVar*' on almost all their components to assign a name to the widget. If this attribute is not defined, then it follows the given algorithm to assign the widget name, which could then be used to access the widget javascript object.

- *widgetVar Naming Algorithm*: "widget_" + JSF clientId – If the clientId has any '-' (dash), it will be replaced with '_' (underscore). If the clientId has any Naming Container Separator (default is ':'), it will be replaced with '_' (underscore)

PF provides a short hand global javascript variable name 'PF' to access the widget javascript object, by a simple reference such as

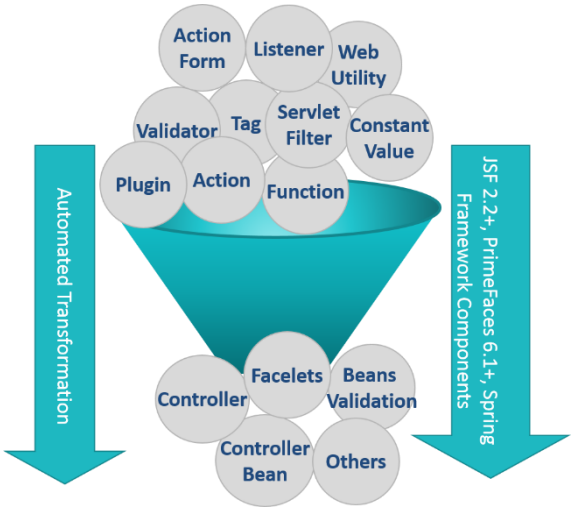
```
var pfWidget = PF('widgetVar');
```

PF also sets a global javascript variable name 'PrimeFaces', that can be used to execute various type of PF related javascript functions and getting hold of a widget, through the code defined below.

```
Var pfWidget = PrimeFaces.widgets.<<actual widgetVar>>;
var pfWidget = PrimeFaces.widgets['<<actual widgetVar>>'];
```

VI. AUTOMATED TRANSFORMATION

An enterprise system that might have been built on Apache Struts 1.x,which has around 100,000 source lines of code (SLOC) for their web components, might benefit by automatically transforming their system through a tool-based approach. The tool can be home grown or commercial. A possible automated transformation approach could be depicted as follows.



VII. CONCLUSION

This paper presents anapproach on how to transform an enterprise system built on Apache Struts 1.x based on open source framework,which has reached its end of life (EOL),into a more mature, mobile friendly, HTML 5 features while incorporating the new features of JEE 7 (+) platform. This paper has touched upon the main structure and components of Struts based system and how to transform it to a JSF based components built on modern technologies such as CDI, BV, PrimeFaces and OmniFaces.

REFERENCES

[1] Apache Struts 1 End of Life Announcement - https://struts.apache.org/struts1eol-announcement.html
[2] CDI 1.1 API - https://docs.jboss.org/cdi/api/1.1/
[3] Java Server Faces Specifications - https://javaee.github.io/javaserverfaces-spec/
[4] Jakarta Enterprise Edition (JEE) - https://jakarta.ee/
[5] JEE 7 Specification - https://jcp.org/en/jsr/detail?id=342
[6] JEE 8 Specification - https://jcp.org/en/jsr/detail?id=366
[7] PrimeFaces - https://www.primefaces.org/#primefaces
[8] OmniFaces - http://omnifaces.org/