PrimeFaces

Jaroslav Dytrych

Faculty of Information Technology Brno University of Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole

dytrych@fit.vutbr.cz

21 October 2020

PrimeFaces

Introduction

- JSF doesn't provide rich set of components
  - It is left for 3rd party libraries
- PrimeFaces
  - rich set of components
  - uses JQuery library for custom components
  - AJAX support (based on JSF 2.0)
  - push support via Atmosphere framework (WebSocket/Comet)
  - one-jar library, no configuration nor dependencies
  - lots of built-in themes, visual theme designer tool (ThemeRoller)
    - `https://jqueryui.com/themeroller/`
  - extensive documentation
  - XHTML facelets on client combined with Java on the server side

Content

- PrimeFaces
  - Theming concept
  - Inputs and selects
  - Client side validations
  - Panels
  - Data iteration components
  - Menus
  - Dialog framework
  - Working with files and images
  - Drag & Drop
  - Charts
  - Push
  - RequestContext

Theming concept

- ThemeRoller CSS Framework
  - over 30 pre-designed themes
- Configuration (`web.xml`)

```
<context-param>
    <param-name>primefaces.THEME</param-name>
    <param-value>aristo</param-value>
</context-param>
```

- May be dynamic

```
<context-param>
    <param-name>primefaces.THEME</param-name>
    <param-value>#{loggedInUser.preferences.theme}</param-value>
</context-param>
```

Custom theme

- Custom theme must be present in one `.jar` file.
- mandatory structure

```
.jar
  – META-INF
      – resources
          – primefaces-yourtheme
              – theme.css
              – images
```

- Image adressing
  - `url("images/my_image.png") must be changed to`
  - `url("#{resource['primefaces-yourtheme:images/my_image']}")`

Predefined selectors

Selector	Description
.ui-widget	All PrimeFaces components
.ui-widget-header	Header section of a component
.ui-widget-content	Content section of a component
.ui-state-default	Default class of a clickable
.ui-state-hover	Class applied when cursor is over widget
.ui-state-active	When clickable is activated
.ui-state-disabled	Disabled elements
.ui-state-highlight	Highlighted elements
.ui-icon	An element to represent an icon

Inputs and selects

- Input mask
  - minimizes the chances for the user to input incorrect data
    ```
    <p:inputMask value="#{maskController.phone}"
                 mask="(999) 999-999-999"/>
    ```
  - a kind of regular expressions
  - 9 is used as a pattern for 0 – 9
- Input language
  - kind of regular expressions for validating input
  - asterisk for multiple occurrence
  - question mark for optional occurrence

```
<p:inputMask value="#{inputMaskController.productKey}"
             mask="a*-999-a999" />
```

Inputs and selects

- Autocomplete
  - method `complete` takes a string and returns a `List<String>`

```
<p:autoComplete id="simple" value="#{autoCompleteController.txt1}"
                completeMethod="#{autoCompleteController.complete}" />
```

- Autocomplete event

```
<p:autoComplete value="#{autoCompleteController.txt1}"
                completeMethod="#{autoCompleteController.complete}">
    <p:ajax event="itemSelect"
            listener="#{autoCompleteController.handleSelect}"
            update="messages" />
</p:autoComplete>

public void handleSelect(SelectEvent event) {
    Object selectedObject = event.getObject();
    MessageUtil.addInfoMessage("selected.object", selectedObject);
}
```

- Every input component can fire appropriate AJAX events when they occur.

Example PFInput

Other input elements

- `InputTextArea`
  - events/attributes: `onkeyup`, `onfocus`, `onblur`, ...
- `TextEditor`
  - rich text editing features (`https://quilljs.com/`)
- `SelectManyCheckBox`
  - used to choose multiple items from a collection
- Calendars
  - multiple display modes and effects
- `Spinner`
  - boundaries
- `Slider`
  - it is possible to set min/max value, step, range, ...
  - vertical or horizontal
- ...

Examples PFEvents, PFBasicLayout, PFCalendar

Partial processing

- Partial processing allows updating JSF components with AJAX.
- Partial processing speeds up large form processing.
- Partial rendering defines elements to be updated.

```
<h:form id="myform">
    <p:commandButton value="Update" update="myform:display" />
    <h:outputText id="display" value="#{bean.value}"/>
</h:form>
```

- Partial validations
    - may prevent unwanted validations

```
<h:form>
    <h:selectOneMenu id="cities" value="#{bean.city}">
        <f:selectItems value="#{bean.cityChoices}" />
        <p:ajax actionListener="#{bean.populateSuburbs}"
                event="change" update="suburbs" process="@this"/>
    </h:selectOneMenu>
    ...
</h:form>
```

Example PrimePartialProcessing

Partial processing

- Search expression framework

Keyword	Type	Description
@this	Standard	Current component
@all	Standard	Whole view
@form	Standard	Closest ancestor form
@none	Standard	No component
@namingcontainer	PrimeFaces	Closest ancestor naming container
@parent	PrimeFaces	Parent of the current component
@composite	PrimeFaces	Closest composite component ancestor
@child(n)	PrimeFaces	Nth child
@previous	PrimeFaces	Previous sibling
@next	PrimeFaces	Next sibling
@widgetVar(name)	PrimeFaces	Component with given widget variable

Client side validations

- Validations must be compatible with server side implementation.
- Conversion and validation happens at client side.
- Partial process&update support for AJAX.
- i18n support along with component specific messages.
- Client side renderers for message components.
- Easy to write custom client converters and validators.
- Global or component based enable/disable.
- Advanced bean validation integration.
- Little footprint using HTML5.

Validations

- Client side validations are disabled by default, has to be enabled in configuration

```
<context-param>
  <param-name>primefaces.CLIENT_SIDE_VALIDATION</param-name>
  <param-value>true</param-value>
 </context-param>
```

- Non-AJAX
  - In non-AJAX case, all visible and editable input components in the form are validated and message components must be placed inside the form.
- AJAX
  - partial processing and updates
- Custom validation
  - implementing client validation interface
    - method `validate()`

Example PFValidations

Validations

- Bean validation
  - constraints via annotations

```
<h:form>
    <p:growl />
    <h:panelGrid>
        <h:outputLabel for="name" value="Name:" />
        <p:inputText id="name" value="#{bean.name}" label="Name"/>
        <p:message for="name" />
        <h:outputLabel for="age" value="Age: (@Min(10) @Max(20))" />
        <p:inputText id="age" value="#{bean.age}" label="Age"/>
        <p:message for="age" />
    </h:panelGrid>
    <p:commandButton value="Save" validateClient="false" ajax="false" />
</h:form>

public class Bean {
    @Size(min=2,max=5)
    private String name;
    @Min(10) @Max(20)
    private Integer age;
}
```

  - growl is used for messages (in the top right corner)

Messages

- Messages components are used to display FacesMessages.
    - Severity: Info, Warn, Error or Fatal.
    - Messages can indicate errors in the forms.

```
<p:messages id="messages" showDetail="true" autoUpdate="true"
            closable="true" />
...
<p:outputLabel for="txt" value="Text:" />
<p:inputText id="txt" required="true" />
<p:message for="txt" display="text" />
```

```
FacesContext.getCurrentInstance().addMessage(null,
    new FacesMessage(FacesMessage.SEVERITY_FATAL, "Fatal!",
    "System Error"));
```

Examples PFMessages

Panels

- Panels serves as containers for storing of other widgets.
- Panel is a generic component.
  - toggling
  - closing
  - built-in pop-up menu
  - AJAX listeners
- Panel grid
  - support for colspan and rowspan.
- Dynamic content loading
  - Tabs can be lazily loaded based on a value of underlying JavaBean.
- Dynamic tabbing
  - `AccordionPanel`

Example PFAccordion

Panels

- Overflow content
  - `ScrollPanel`
- Buttons grouping
  - toolbars, separators
- Draggable widgets
  - `DashBoard` panel
  - grid with row and columns constraints
- Full Page layout
  - North, West, Center, East, South
- Element layout
  - at element level
  - styled with CSS
- Nested layouts
- Panels can fire appropriate events
  - `close, toggle, resize`

```
<p:ajax event="close" listener="#{panelView.onClose}"
        update="msgs" />
```

Examples PFFullPageLayout, PFNestedLayout

Data iteration components

- Data iteration components are usually data tables or trees.
- Selection
  - selection mode (`single` or `multiple`)

```
<p:dataTable id="multipleSelectionCheckbox" var="car"
             value="#{dataTableController.cars}"
             rowKey="#{car.name}"
             selection="#{dataTableController.selectedCars}">
    <p:column selectionMode="multiple"/>
    ...
</p:dataTable>
```

  - property listeners
    - Selected object is referenced as a variable and can be passed to underlying Java method.

```
<f:setPropertyActionListener value="#{car}"
    target="#{dataTableController.selectedCar}" />
```

Example PFDataTable

Data iteration components

- Sorting and filtering in DataTable
  - Sorting

```
<p:dataTable id="sorting" var="car"
             value="#{dataTableController.cars}">
    <p:columnheaderText="Year" sortBy="#{car.year}">
    <h:outputText value="#{car.year}" />
```

  - Filtering
    - displays filter text fields
    - user filters the data
    - all fields can be searched

```
<p:dataTable id="filtering" var="car"
             value="#{dataTableController.cars}">
    <p:column headerText="Year" filterBy="#{car.year}">
        <h:outputText value="#{car.year}" />
    </p:column>
    <p:column headerText="Name" filterBy="#{car.name}">
        <h:outputText value="#{car.name}" />
    </p:column>
</p:dataTable>
```

Data iteration components

- In cell editing
  - AJAX events

```
<p:ajax event="rowEdit"
        listener="#{dataTableController.onEdit}"
        update=":form:growl" />
<p:ajax event="rowEditCancel"
        listener="#{dataTableController.onCancel}"
        update=":form:growl" />
```

- Lazy models – handling lots of records
  - supports pagination
  - `org.primefaces.LazyDataModel`
  - Programmer must implement `load`, `getRowData` and `getRowKey` methods.

```
<p:dataTable id="lazyModel" var="car"
             value="#{lazyDataTableController.lazyModel}"
             paginatorTemplate="{RowsPerPageDropdown} {FirstPageLink}
                 {PreviousPageLink} {CurrentPageReport} {NextPageLink}
                 {LastPageLink}"
             paginator="true" rows="10" lazy="true">
```

Example PFTableCell

Data iteration components

- Trees and TreeTables
  - Events
    - collapse, expand, select, unselect
- Context menu support

```
<p:contextMenu for="withContextMenu" nodeType="node">
    <p:menuitem value="View" update="dialogPanel"
                icon="ui-icon-search"
                oncomplete="nodeDialog.show()"/>
</p:contextMenu>
<p:contextMenu for="withContextMenu" nodeType="leaf">
    <p:menuitem value="View"
                update="dialogPanel" icon="ui-icon-search"
                oncomplete="nodeDialog.show()"/>
    <p:menuitem value="Delete"
                update="withContextMenu" icon="ui-icon-close"
                actionListener="#{treeDataController.deleteNode}"/>
</p:contextMenu>
```

Menus

- Menu positioning
  - static
    - displayed in page by default
  - dynamic
    - overlay, not displayed by default
    - defines trigger button, position relative to that button
- Programmatic menu
  - Menu can be defined also in Java
    `<p:menu model="#{programmaticMenuController.model}"/>`
  - Model object returns constructed menu.
- Context menu

```
<p:contextMenu for="fileSystem">
    <p:menuitem value="View" update="growl"
                actionListener="#{contextMenuController.viewNode}"
                icon="ui-icon-search"/>
    <p:menuitem value="Delete" update="fileSystem"
                actionListener="#{contextMenuController.deleteNode}"
                icon="ui-icon-close"/>
</p:contextMenu>
```

Example PFMenu, PFContextMenu

Menus

- Other menus
  - `Menubar`
    - displays root items horizontally and nested items as tiered
    - for static menus
  - `MegaMenu`
    - multi-column menu
    - displays submenus of root items together
  - `TieredMenu`
    - submenus in nested overlays
  - `PanelMenu`
    - hybrid of accordion-tree
  - `SlideMenu`
    - displays nested submenus with a slide animation
  - `SelectCheckBoxMenu`
    - menu with checkboxes which are on or off

Example PFMenuBar

Dialogs

- Simple dialogs
  - `<p:dialog` ...
  - Yes|No questions, notifications, asking for input
- Dialog framework
  - opens an external XHTML page in a dialog that is generated
- Dialogs requires configuration in `faces-config.xml`

```
<application>
    <action-listener>
        org.primefaces.application.DialogActionListener
    </action-listener>
    <navigation-handler>
        org.primefaces.application.DialogNavigationHandler
    </navigation-handler>
    <view-handler>
        org.primefaces.application.DialogViewHandler
    </view-handler>
</application>
```

Examples PFDialogs, PFCustomDialog

Files and images

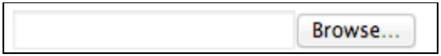
- `FileUpload` component
  - equivalent to HTML `<input type="file">`
  - HTML 5 powered UI, such as Drag & Drop
- Approaches
  - Native
    - works since JSF 2.2 – Servlet Part API
  - Commons
    - requires configuration
    - may specify size threshold, upload directory (`init-param`), ...

```
<filter>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <filter-class>
        org.primefaces.webapp.filter.FileUploadFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>
```

Files and images

- Two file upload modes
  - Simple

```
<h:form enctype="multipart/form-data">
    <p:fileUpload value="#{fileController.file}" mode="simple" />
    <p:commandButton value="Submit" ajax="false"/>
</h:form>
```

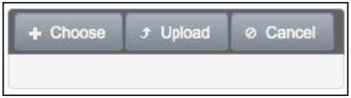


  - Advanced
    - specifies upload handler

```
<p:fileUpload mode="advanced"
    fileUploadListener="#{fileController.handleFileUpload}" />
```

    - may limit maximum number of files to be uploaded, file type etc.

```
public void handleFileUpload(FileUploadEvent event) {
    UploadedFile file = event.getFile();
    MessageUtil.addInfoMessage("upload.successful",
        file.getFileName() + " is uploaded.");
}
```



Examples PFFile, PFMultiUpload

Files and images

- File download
  - bean must return streamed content

```
<p:commandButton value="Download" ajax="false">
    <p:fileDownload value="#{fileController.file}" />
</p:commandButton>
```

- Status of upload/download is monitored by JavaScript.
- Galleria widget for multiple images

```
<p:galleria value="#{galleriaController.cars}" var="car">
    <p:graphicImage
        value="/resources/images/autocomplete/#{car.name}.png"/>
</p:galleria>
```

Example PFDownload

PrettyFaces

- PrettyFaces is an OpenSource URL-rewriting library with enhanced support for JavaServer Faces.
- Enables creation of bookmarkable, pretty URLs (Search Engine Optimization friendly).
- Maven dependency (or `.zip` distribution).
- Workflow:
  - Add PrettyFaces to your `pom.xml`
  - Create `pretty-config.xml`

```
<url-mapping id="login">
    <pattern value="/login" />
    <view-id value="/user/login.jsp" />
</url-mapping>

<!-- Map "/user/#{username}"
  to the URL "/user/view.xhtml?username=value" -->
<url-mapping id="view-user">
    <pattern value="/user/#{username}" />
    <view-id value="/user/view.xhtml" />
</url-mapping>
```

  - Run your application.

PrimeFaces with PrettyFaces (1/2)

- PrettyFaces breaks PrimeFaces file upload.
- Prerequisites
  - `commons-fileupload` and `commons-io` are present in the webapp's runtime classpath (`/WEB-INF/lib`)
  - The `FileUploadFilter` is mapped on the exact `<servlet-name>` of the FacesServlet as is been definied in your `web.xml`.
  - The enctype of the `<h:form>` needs to be set to `multipart/form-data`.
  - In simple file upload with `mode="simple"`, AJAX must be disabled on any PrimeFaces command buttons/links by `ajax="false"`.

PrimeFaces with PrettyFaces (2/2)

- Solution (`web.xml`):

```
<filter>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <filter-class>
        org.primefaces.webapp.filter.FileUploadFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>PrimeFaces FileUpload Filter</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

Drag & Drop

- Making panel draggable

```
<p:panel id="pnl" header="Draggable panel with default settings">
    <h:outputText value="Drag me around"/>
</p:panel>
<p:draggable for="pnl"/>
```

- Draggable restrictions
  - Horizontal `<p:draggable for="hpnl" axis="x"/>`
  - Vertical `<p:draggable for="vpnl" axis="y"/>`
  - Grid `<p:draggable for="gpnl" grid="40,50"/>`
  - Boundary `<p:draggable for="pic" containment="parent"/>`

- Drag & Drop may be used in AJAX requests,
- can be integrated with data iteration components.

Drag & Drop

- Defining draggable targets
  - Client-side callback `onDrop`

```
<h:panelGroup id="drop" layout="block" styleClass="ui-widget-content"
              style="height:150px;width:300px;">
    <p class="ui-widget-header" style="margin:0;padding:5px;">
        Drop here
    </p>
    <p:droppable onDrop="handleDrop" tolerance="fit"/>
</h:panelGroup>
```

- Dropping restrictions
  - defining tolerance and acceptance
  - Tolerance specifies which mode to use for testing if a draggable component is over a droppable.
    - Four types of tolerance – `fit`, `intersect`, `pointer`, `touch`
  - Acceptance defines scope atributes, dropable must have same scope as dragable if Drag & Drop is to be applied.
  - Scope is some sort of string id

```
<p:droppable onDrop="handleDrop" scope="dnd"/>
<p:draggable scope="dnd"/>
```

Charts

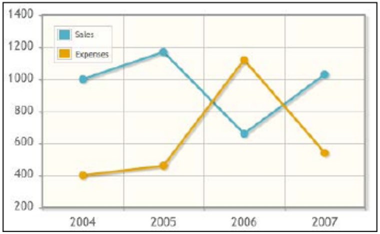
- PrimeFaces provides simple API for displaying various types of Charts
- Client-side chart refers to Chart model value defined on the server.
- Types
  - Area
  - Bar
  - Line
  - Bubble
  - Donut
  - Pie
  - ...
- Appropriate model value has to be returned from JavaBean

```
<p:lineChart value="#{chartController.model}" style="height:250px" />
```

Charts

- Facelet defines legends, axis values etc.
- Java implementation of a model

```
CartesianChartModel model = new CartesianChartModel();
ChartSeries sales = new ChartSeries();
sales.setLabel("Sales");
sales.set("2004", 1000);
sales.set("2005", 1170);
sales.set("2006", 660);
sales.set("2007", 1030);
ChartSeries expenses = new ChartSeries();
expenses.setLabel("Expenses");
expenses.set("2004", 400);
expenses.set("2005", 460);
expenses.set("2006", 1120);
expenses.set("2007", 540);
model.addSeries(sales);
model.addSeries(expenses);
```



RemoteCommand

- RemoteCommand provides a simple way how to execute backing bean methods with JavaScript.

```
<h:form>
    <p:remoteCommand name="rc" update="msgs"
                     actionListener="#{remoteCommandView.execute}" />
    <p:growl id="msgs" showDetail="true" />
    <p:commandButton type="button" onclick="rc()" value="Execute"
                     icon="ui-icon-refresh" />
</h:form>
```

- can be used also for partial processing of the form

```
<h:form id="form">
    <p:remoteCommand name="updateList" update="users" process="@this" />
...
function handleMessage(message) {
    ...
    updateList();
}
```

Push

- Atmosphere framework is used for sending asynchronous messages from the server to the client.
- Requires special configuration (`web.xml`)

```
<servlet>
    <servlet-name>Push Servlet</servlet-name>
    <servlet-class>org.primefaces.push.PushServlet</servlet-class>
    <async-supported>true</async-supported>
</servlet>
<servlet-mapping>
    <servlet-name>Push Servlet</servlet-name>
    <url-pattern>/primepush/*</url-pattern>
</servlet-mapping>
```

- Uses annotations for defining push endpoints and message callbacks.

Push

- `@PushEndpoint`
  - A class annotated with this annotation defines push channel, through which the server can contact the client.
- `@OnMessage`
  - When data are ready to be delivered, method annotated with this annotation will be called.
- Connection lifecycle annotations
  - `@OnOpen`
  - `@OnClose`
- `@PathParam`
  - parameters in path in URI

```
@PushEndpoint("/somepath/{room}/{user}")
@Singleton
public class ChatResource {
    @PathParam("room")
    private String room;
    @PathParam("user")
    private String username;
    ...
```

Push

- API
  - `RemoteEndPoint`
    - represents client-side browser
  - `EventBus`
    - class for interacting with Push endpoints
    - uses Pub-Sub and Point-to-Point messaging domains

      ```
      EventBus eventBus =
          EventBusFactory.getDefault().eventBus();
      eventBus.publish("/counter", "Some data");
      ```
- Encoders and decoders
  - has to be used when broadcasting a value

```
@PushEndpoint("/counter")
public class CounterResource {
    @OnMessage(encoders = {JSONEncoder.class})
    public String onMessage(String count) {
        return count;
    }
}
```

Push

- Client side
  - has to declare socket, through which it can accept the data.

```
<h:form id="form">
    <h:outputText id="out" value="#{globalCounter.count}" />
    <p:commandButton value="Click"
                     actionListener="#{globalCounter.increment}" />
</h:form>

<p:socket channel="/counter">
    <p:ajax event="message" update="form:out" />
</p:socket>
```

  - socket defines a channel
  - often convenient to use JavaScript

```
<p:socket onMessage="handleMessage" channel="/notify" />
<script type="text/javascript">
    function handleMessage(facesmessage) {
        facesmessage.severity = 'info';
        PF('growl').show([facesmessage]);
    }
</script>
```

Example PFChat (JBoss)

RequestContext

- Update component(s) programmatically.
    - dynamic rendering
- Execute JavaScript from beans.

```
if (!FacesContext.getCurrentInstance().isPostback()) {
  RequestContext.getCurrentInstance()
    .execute("alert('This onload script is added from backing bean.')");
}
```

- Add AJAX callback parameters.
- Scroll to a specific component after AJAX update.

Example PFRequestContext

References

- http://www.primefaces.org/showcase/
- http://primefaces.org/
- http://www.ocpsoft.org/prettyfaces/
- http://blog.hatemalimam.com/using-prettyfaces-with-primefaces-upload/
- https://jqueryui.com/themeroller/

Thank you for your attention!