JSF Avatar

13 June 2006

Ed Burns

Senior Staff Engineer

Enterprise Java Platforms

Agenda

- You already know what avatar is...
- What's changed since last update (on 19 April)?
- Current Thinking
- Roadmap

What's changed since last update?

- Ed and Jacob working independently, collaborating on concept level
- Each had separate implementations leading into JavaOne
- Ed's approach:
  - ajaxZone for each dynamically updated area. This is the main API to avatar
  - User provides JavaScript functions to fill well-defined contract to construct the AJAX request
  - Avatar JS library calls moderately complex user functions, builds request, handles response
  - Avatar JS library not exposed to users.
- Jacob's approach:
  - No ajaxZone, User must point to the sub-tree manually using clientIds
  - Avatar JS library exposed to user. This is the main API to avatar
  - User-level JS functions tend to be very simple, mostly defined in-line.

Current Thinking: API

- Two paths, both valid
  - > Expose Jacob's JavaScript library
    - > `<h:inputText id=”email” value=”#{employee.email}”`
      `onblur=”new Faces.Event(this, { update: this.name });” />`
    - > `<a href=”#” id=”click”>click</a>`
    - > `<script>new Faces.Command($(click), 'mousedown',`
    - > `{ encode: 'catalog' });`
  - > Make ajaxZone use that library to get its work done.
- Avatar JS library
  - > Exposes Faces.Event and Faces.Command
  - > Responsible for initiating AJAX requests, incorporating responses into current DOM.

Current Thinking: Request Headers

- Request headers convey metadata, indicate AJAXiness
- Are added to the Ajax request by the avatar js library.
  - com.sun.faces.Async: true
  - com.sun.faces.Subtrees: form:subview1,form:subview2
  - (optional) com.sun.faces.RunThru: <PHASE_ID>
  - (optional) com.sun.faces.lifecycle.<PHASE_ID>: form:subview1,form:subview2

Current Thinking: Lifecycle

- Custom JSF Lifecycle Implementation. Decorates the default Lifecycle impl.
- All requests with header com.sun.faces.Async: true treated as AJAX request. Otherwise, default Lifecycle impl is used.

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <init-param>
    <param-name>javax.faces.LIFECYCLE_ID</param-name>
    <param-value>com.sun.faces.lifecycle.AJAX</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern><url-pattern>*.faces</url-pattern>
</servlet-mapping>
```

Current Thinking: Lifecycle

- No custom UIViewRoot
- Use the custom lifecycle instead of the default one.
- Examines request headers and takes action accordingly
- Makes use of `invokeOnComponent()` to ensure component context when running a lifecycle phase on a particular subtree.
- Renders the XML for the AJAX response
- Calls state save and restore APIs

Roadmap

- Integrating Ed and Jacob's approach into one codebase.
- Adding new ideas
- Plan to be done initial merge by 30 June
- Would like to get Exadel's ajax4jsf project on board after initial merge.