JSF And Ajax : Past Present And Future

Andy Schwartz
Oracle Corporation

Roger Kitain
Sun Microsystems

Agenda

- JSF 101
- JSF and Ajax In Action
- JSF 2.0 Ajax

JSF 101

What is JSF?

- Component-centric web framework
- Page contents defined declaratively
  - JSP, Facelets
- Standard request processing lifecycle
- Managed bean system
- EL binding
- Navigation system
- Conversion/validation system
- Event system

Sample: JSF “Hello, World!”

```
<h:form>
  <h:inputText value="#{user.name}"/>
  <h:commandButton value="Greet"/>
  <h:outputText value="Hello, #{user.name}!"
</h:form>
```

Sample: JSF Managed Bean

```
public class User
{
  public String getName()
  {
    return _name;
  }

  public void setName(String name)
  {
    _name = name;
  }

  private String _name;
}
```

Sample: JSF Configuration

```
<managed-bean>
  <managed-bean-name>user</managed-bean-name>
  <managed-bean-class>pojsf.User</managed-bean-clas
  <managed-bean-scope>request</managed-bean-scope
</managed-bean>
```

Initial Page Lifecycle

- Build view
- Render view
- Save state

Postback Lifecycle

- Restore view from state
- Decode request
- Convert/validate data
- Update model
- Invoke application
- Render response
- Save state

Components and Renderers

- UIComponent defines logical representation
  - Component type, properties, parent/children
- Renderer defines physical representation
  - HTML, XHTML, other markup
- Separation provides flexibility
  - Alternate RenderKits

JSF and Ajax In Action

Getting Started With Ajax: Shale Remoting

- Remote access to resources
  - Class path
  - Web context
  - Managed beans
- Resources exposed via URL conventions
- Managed bean remote access provides basis for simple Ajax solution
- URL pattern:
  - /dynamic/<bean name>/<method name>

Shale Remoting Sample: Client-Side

```
<h:inputText value="#{user.name}"
             onblur="validateName(this.value);"/>

function validateName(name) {
  new Ajax.Request(
    "dynamic/user/validateName.jsf",
    {
      method: "post",
      parameters: "name=" + name,
      onComplete: doSomething
    }
  );
}
```

Shale Remoting Sample: Bean-Side

```
public void validateName() {
    // Use standard JSF APIs to extract name parameter
    String name = _getNameFromRequest();

    // Check to see whether name is valid
    if (!_isNameValid()) {
      // If not, send back a response to let the
      // client know.
      _writeInvalidResponse();
    }
  }
```

Shale Remoting Usage

- No JavaScript API provided
  - Use Prototype, or your favorite library
  - Handle your own DOM updates
- Minimal integration with JSF lifecycle
  - No access to component tree, only managed beans
- Good for simple Ajax/managed bean cases, such as validation
  - Don’t forget to re-validate on the server

Shale Remoting Vital Info

- Open Source
- Hosted at Apache (Apache Shale project)
- License: Apache Software License 2.0
- Fairly quiet in terms of developer/user activity

Getting In Your Faces: Dynamic Faces

- Managed bean remoting a good start, but does not leverage components, renderers
- Dynamic Faces provides tighter JSF integration
- JavaScript API for posting back into Faces
- Faces lifecycle integration for producing “partial” responses
  - Leverages invokeOnComponent()
- Ajax-side response handling for updating the DOM
- “No JavaScript Needed” solution available with “ajaxZone” tag.

Dynamic Faces JavaScript API

- DynaFaces.fireAjaxTransaction(domElement, options)
  - domElement - domElement which triggers the Ajax request
  - options - JS object with well known properties
- "render" option allows specific subtrees to be targeted for re-render
- “execute” option constrains other phases
- Various other options (asynchronous, immediate, inputs, methodName, etc...)

Dynamic Faces Sample: “Hello, World!”

```
<h:form>
   <h:inputText value="#{user.name}"/>
   <h:commandButton id=”submit” value="Greet"
     onclick=”DynaFaces.fireAjaxtransaction(this,
     {execute:'submit', render:'greeting'}); return false;”/>
   <h:outputText value="Hello, #{user.name}!"
                        id="greeting“ />
 </h:form>

<jsfExt:ajaxZone>
<h:form>
   <h:inputText value="#{user.name}"/>
   <h:commandButton id=”submit” value="Greet" />
   <h:outputText value="Hello, #{user.name}!"
        rendered="#{user.name != null}" id="greeting“ />
</h:form>
</jsfExt:ajaxZone>
```

Dynamic Faces Response Payload

```
<partial-response>
  <components>
    <render id=”form:table”>
      <markup>
        <![CDATA[.....rendered content....]]>
      </markup>
    </render>
    <render...
     ...
    </render>
  </components>
</partial-response>
```

Dynamic Faces Vital Info

- Open Source
- Hosted at java.net (JSF Extensions project)
- License: CDDL 1.0
- Fairly quiet in terms of developer/user activity, though interest from JSF 2.0 EG

Getting Declarative: Ajax4JSF

- fireAjaxTransaction() provides clean entry point into Faces lifecycle, but requires JS code
- Ajax4JSF tackles the same problem, but with a declarative (tag-based) solution
- a4j:support allows Ajax capabilities to attached to existing components

Ajax4JSF Sample: “Hello, World!”

```
<h:form>
  <h:inputText value="#{user.name}"/>
  <h:commandButton value="Greet">
      <a4j:support event="onclick" reRender="greeting"/
  </h:commandButton>
  <h:outputText value="Hello, #{user.name}!"
                        id="greeting“ />
</h:form>
```

Ajax4JSF a4j:support

- Provides fine-grained control over Ajax request/response
  - bypassUpdates
  - disableDefault
  - eventsQueue
  - focus
  - oncomplete
  - onsubmit
  - reRender
  - requestDelay
  - timeout

Ajax4JSF Response Payload

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head></head>
  <body>
    <span id="foo">a</span>
    <meta name="Ajax-Update-Ids" content="foo"/>
    <meta id="Ajax-Response" name="Ajax-Response"
          content="true"/>
  </body>
</html>
```

Ajax4JSF Ajax-Enabled Components

- Common Ajax operations simplified with specialized components
  - a4j:commandButton
  - a4j:commandLink
  - a4j:poll

Ajax4JSF Sample Revisited: “Hello, World!”

```
<h:form>
   <h:inputText value="#{user.name}"/>
   <a4j:commandButton value="Greet"
                reRender="greeting"/>
   <h:outputText value="Hello, #{user.name}!"
                id="greeting“ />
 </h:form>
```

Ajax4JSF Vital Info

- Open Source
- Originally Developed By Exadel
- Donated to Jboss along with RichFaces
- License: LGPL
- Very active development/user communities, in part due to RichFaces

Another Approach: Apache Trinidad

- Like Ajax4JSF, Trinidad provides Ajax-enabled command components:
  - tr:commandButton
  - tr:commandLink
- Also provides Ajax-enabled input, select components as well
  - tr:inputText
  - tr:selectOneChoice
  - Many more…

Apache Trinidad: Enabling Ajax Requests

- Unlike Ajax4JSF, Ajax requests must be explicitly enabled
- “partialSubmit” for command components
  - When true, Ajax-style post back, otherwise, traditional full page post back.
- “autoSubmit” for input, selects
  - When true, Ajax-style post back on change

Apache Trinidad: Partial Targets and Triggers

- Partial “targets” are components that are re-rendered during the Ajax response.
- Partial targets can be specified programmatically
  - PartialPageContext.addPartialTarget()
- Partial targets also specified declaratively via the “partialTriggers” attribute
- When partial “trigger” component fires, target component is included in the response

Apache Trinidad Sample: “Hello, World!”

```
<tr:form>
  <tr:inputText value="#{user.name}"/>
  <tr:commandButton text="Greet"
                    id="greetButton"
                    partialSubmit="true“ />
  <tr:outputText value="Hello, #{user.name}!"
                    partialTriggers="greetButton“ />
</tr:form>
```

Apache Trinidad: Partial Lifecycle

- Trinidad uses ResponseWriter switching
- Full traversal of component tree
- Partial targets content pass through
- Non-partial targets write to /dev/null
- Why not invokeOnComponent()?
  - Single pass
  - Render-time side effects

Apache Trinidad: Response Payload

```
<?xml version="1.0" ?>
<?Tr-XHR-Response-Type ?>
<content action="…">
   <fragment>
      <![CDATA[
        HTML fragment here
      ]]>
   </fragment>
</content>
```

Apache Trinidad: Higher Level Components

- In addition to basic command/input/select components, Trinidad includes higher-level Ajax-enabled components
- Components internally leverage Ajax to update themselves in response to use interaction
- Ajax-enabled by default, no burden on application developer

Apache Trinidad: Ajax-Enabled Components

- inputListOfValues
- panelAccordion
- panelTabbed
- poll
- showDetail
- showDetailHeader
- table
- tree
- treeTable

Apache Trinidad Vital Info

- Open Source
- Originally developed by Oracle
- Donated to Apache MyFaces
- License: Apache Software License, 2.0
- Active developer/user community
  - Development mostly focused on core architecture, less on components

Direct-To-DOM: ICEFaces

- Trinidad/Ajax4JSF encourage declarative Ajax
  - Enable Ajax via markup, not JS code
- ICEFaces simplifies further
- No need to specify partial targets/triggers
- Direct-To-DOM rendering handles this automatically

ICEFaces Sample: “Hello, World!”

```
<ice:form>
  <ice:inputText value="#{user.name}"/>
  <ice:commandButton value="Greet"/>
  <ice:outputText value="Hello, #{user.name}!“/>
</ice:form>
```

ICEFaces Direct-To-DOM Ajax

- Renderers typically stream HTML back to client
- With ICEFaces DOM tree is created on the server
- On Ajax requests, new DOM tree is created and diff'ed against the old tree
- Diffs bundled up into Ajax response
- ICEFaces applies diffs on the client
- No need to specify Ajax dependencies
  - Diff mechanism automatically identifies all changes
- Ajax Push too

ICEFaces Response Payload

```
<updates>
  <update address="client id" tag="tag name">
    <attribute name="..."><![CDATA[...]]></attribute>
    <content>
      <![CDATA[
        Inner HTML content here
      ]]>
    </content>
  </update>
</updates>
```

ICEFaces Vital Info

- Open Source
- Originally developed by ICESoft
- Hosted at ICEFaces.org
- License: Mozilla Public License 1.1
- Very active developer/user community

A Hybrid Approach: ADF Faces

- JSF frameworks are typically server-centric:
  - Server-side components define view
  - All rendering on the server
  - Updating UI requires round-trip
- ADF Faces takes a hybrid approach
  - Server-side components are primary abstraction
  - Initial page contents rendered on server
  - Client-side framework provides additional component/event abstraction

ADF Faces Client-Side Framework

- Client-side component abstraction: AdfUIComponen
- Encapsulates client-side behavior, event handling
- Abstracts application developers from DOM
  - Protects against browser portability issues
  - Component implementations (DOM) can change without breaking app-level code
- Components can update themselves on client
  - Reduces need for (some) round trips
  - Components cannot fully re-render on client

ADF Faces Ajax Support

- Shares API, some implementation Trinidad
  - partialSubmit, autoSubmit, partialTriggers
- Adds lifecycle optimizations
- Adds “streaming” support
- Adds “push” support

ADF Data Visualization Components

- Graphs
- Guages
- Geographic Map
- Pivot Table
- Gantt Chart
- Hierarchy Viewer (coming soon)

ADF Faces Vital Info

- Closed Source
- Free with Oracle app server products
- Licensing fee for non-Oracle app servers
- Active user community (OTN)
- Large-scale use/development within Oracle

And So Much More

- Dojo integration
  - Woodstock, Spring Web Flow 2
- YUI integration
  - YUI4JSF, Mojarra Scales
- Flash/Flex integration
  - Exadel Fiji, jsf-flex
- Google Maps integration
  - GMaps4JSF, ICEFaces, Jboss Rich Faces
- Ajax Push/Comet integration
  - IceFaces, ADF Faces, Rich Faces

And More…

- Backbase
- Netadvantage
- QuipuKit
- RCFaces
- Simplica
- Tobago
- Tomahawk
- WebGalileoFaces
- Etc…

The Good

- Ajax and JSF are a good fit
- Plenty of activity, innovation
- Plenty of choices

The Bad

- Lots of choices: which to choose?
- Lots of implementations: re-inventing the wheel?
- Framework/component compatibility?
- No standard support – 3rd party framework required

JSF 2.0 Ajax

Ingredients of a JSF + Ajax Solution

- Resource Delivery Mechanism
- Partial View Processing
- Partial View Rendering
- Ajaxification Capability
- Ajax Enabled Components

Ingredients of a JSF + Ajax Solution

- Resource Delivery Mechanism
- Partial View Processing
- Partial View Rendering
- Ajaxification Capability

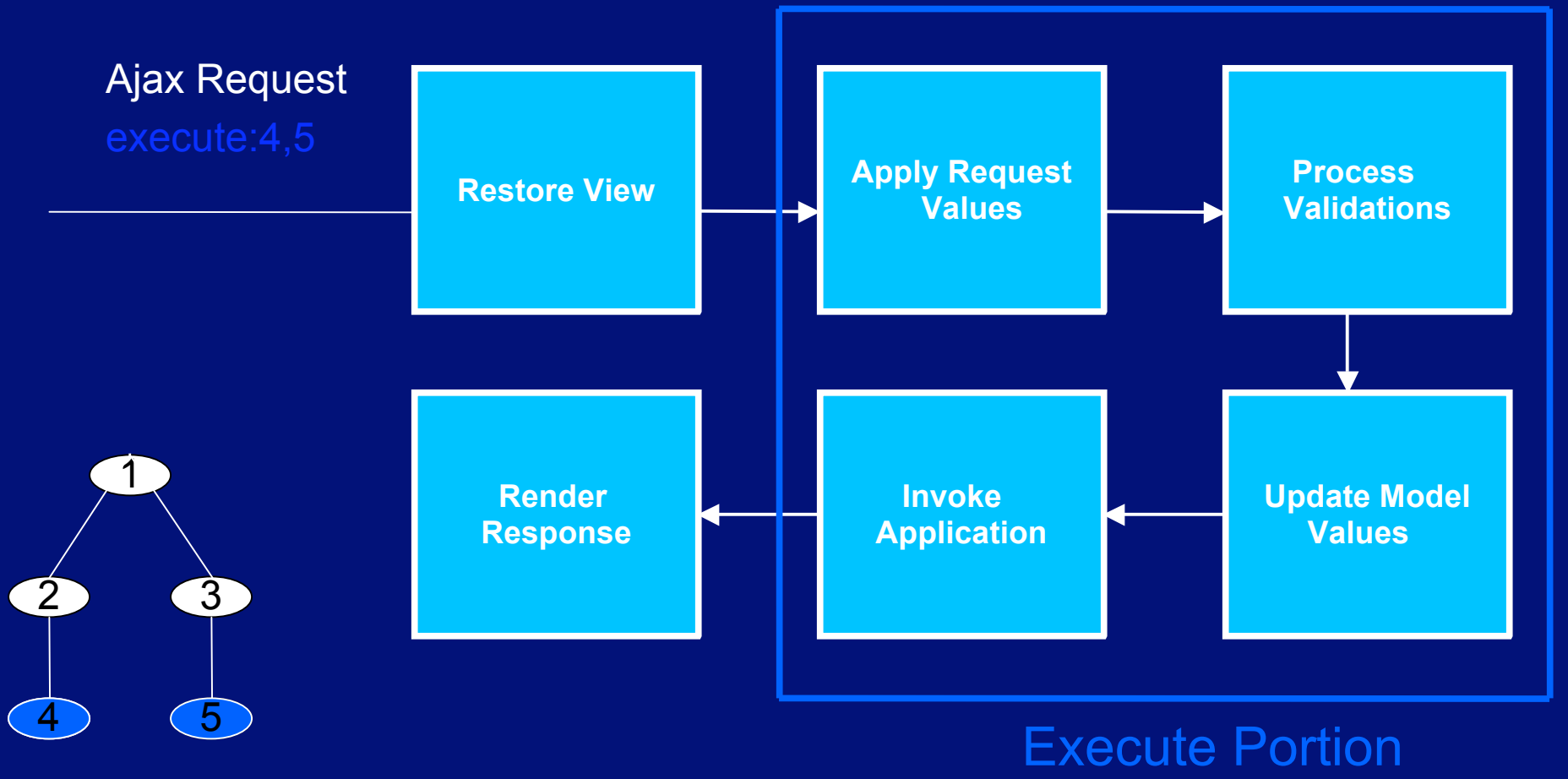
↑ JSF 2.0 Spec

- Ajax Enabled Components

↓ Component Libra

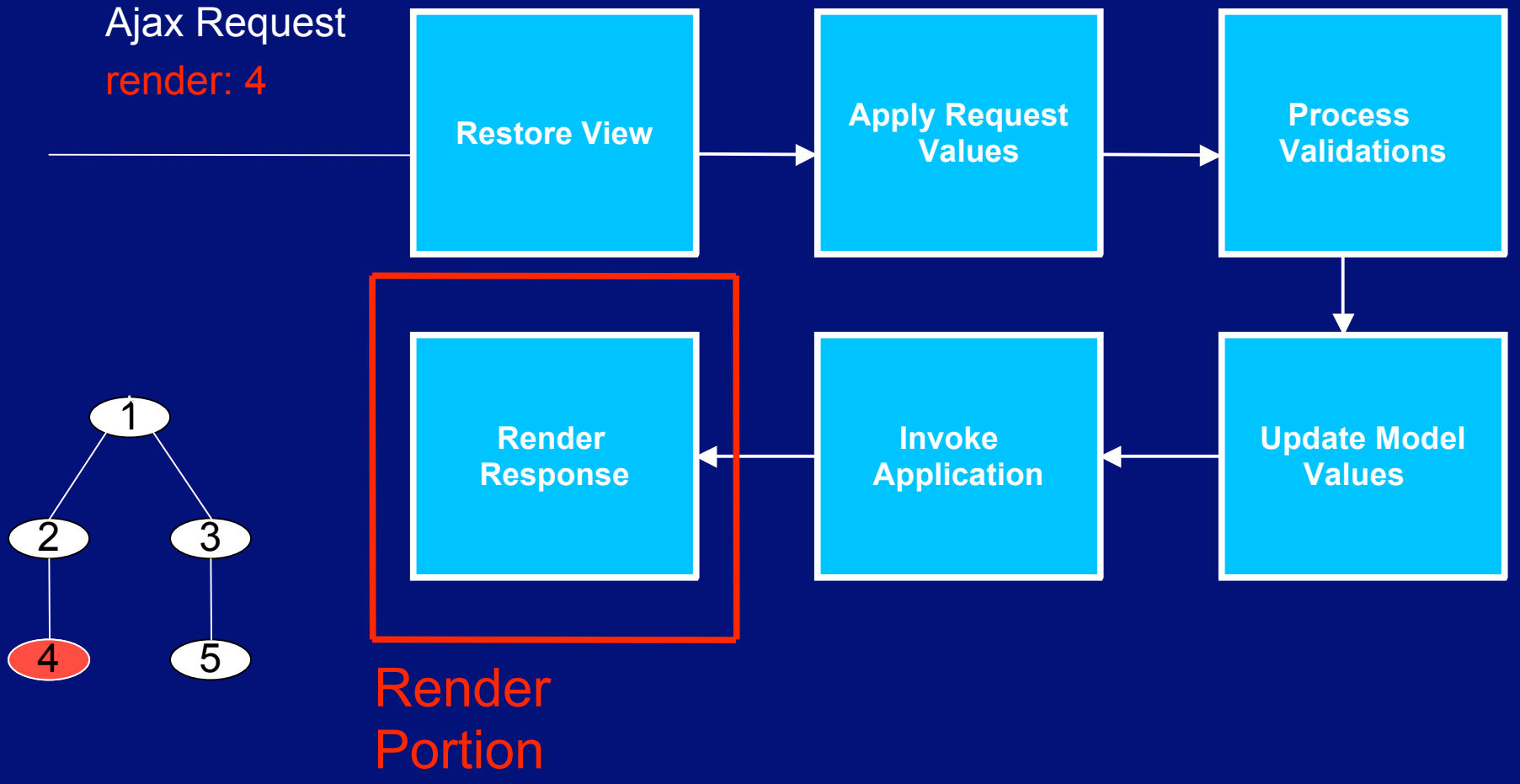
Ingredients of a JSF + Ajax Solution

Partial View Processing



Ingredients of a JSF + Ajax Solution

Partial View Rendering



Ingredients of a JSF + Ajax Solution

Ajaxification Capability

- Provide Ajax capability to existing JSF components without writing JavaScript
- Common approaches:
  - ajaxZone tag : enclose region to ajaxify
  - ajaxSupport tag : nest inside of component to ajaxify

JSF 2.0 Ajax : Details

- Goals:
  - Add common Ajax operations to the standard
  - Fix the “Ajax components from different libraries don't play together” problem

JSF 2.0 Ajax

- JavaScript Namespacing
  - Registered top level namespace with OpenAjax Alliance
  - “faces” property under top level namespace
  - “Ajax” property under “faces” property

JSF 2.0 Ajax

- Public JavaScript functions in “ajax.js”
- Runtime must guarantee delivery of file and namespacing requirements
- Resource loading

JSF 2.0 Ajax : JavaScript API

- javax.faces.Ajax.viewState(form)
  - returns encoded state for the “form” argument
- javax.faces.Ajax.ajaxRequest(element, event, options)
  - sends an asynchronous request to the server
  - all requests go through a client side queue (FIFO)
- javax.faces.Ajax.ajaxResponse(request)
  - receives Ajax response from server and updates the client side DOM
- javax.faces.Ajax.getProjectStage()
  - returns the value of Application.getProjectStage() for the current running application

JSF 2.0 Ajax : Server Side Processing

- PartialViewContext
  - Contains methods/properties that pertain to partial view processing and rendering on a view
  - One per request (accessed via FacesContext)
- UIViewRoot
  - The “driver” for partial processing (decoding) / partial rendering (encoding)
  - UIComponent.invokeOnComponent (JSF 1.2) a key player

JSF 2.0 Ajax : What To Standardize?

- Certainly standardize on areas that will promote component library compatibility
  - Ajax request parameters
  - Sending the Ajax request
  - Ajax response format
    - EG decision: XML as the content type
  - Processing the Ajax response (DOM update)
  - Server side where appropriate

JSF 2.0 Ajax : In Progress

- Declarative Ajax solution
  - promote ease of Ajaxifying components
  - similar to ajaxZone and/or ajaxSupport
- Better error handling
- Ajax Response format
- Portlet support