JavaServer™ Faces Technology, AJAX, and Portlets: It's Easy if You Know How!

Brendan Murray

Software Architect
IBM
http://www.ibm.com

TS-6824

Goal

Why am I here? Why are you here?

Learn what it means to exploit AJAX and JavaServer Faces technology in a Portlet environment and to do it right first time.

Agenda

Introduction and background

Characteristics of each technology

JavaServer Faces technology and Portlet basics

Mixing them all together

The AJAX lifecycle

Summary and questions

Agenda

Introduction and background

Characteristics of each technology

JavaServer Faces technology and Portlet basics

Mixing them all together

The AJAX lifecycle

Summary and questions

Introduction and Background

JavaServer Faces technology, AJAX, and Portlets

- JavaServer Faces Technology
  - Java Community Process℠ (JCP℠) services JSR 252
  - Version 1.2 leverages JavaServer Pages™ (JSP™) 2.1
  - Widely supported: Sun, IBM, BEA, Oracle, etc.
  - Versions
    - Sun's implementation
    - Apache MyFaces (not at v1.2 yet)
  - MVC-2 framework providing a server-based GUI

JSR = Java Specification Request

Introduction and Background

JavaServer Faces technology, AJAX, and Portlets

- Portlets
  - JSR 168 (v2.0 is JSR 286, due later this year)
  - Broad support: Sun, IBM, BEA, Oracle, etc.
  - Open Source version: Apache Pluto
  - Provides an aggregation of web pages on the screen

Introduction and Background

JavaServer Faces technology, AJAX, and Portlets

- Asynchronous JavaScript™ and XML (AJAX)
  - Originally called Remote Scripting
  - Widespread use triggered by Google's applications
  - Technique to create interactive web applications
  - Zero-footprint rich Internet applications

Agenda

Introduction and background

Characteristics of each technology

JavaServer Faces technology and Portlet basics

Mixing them all together

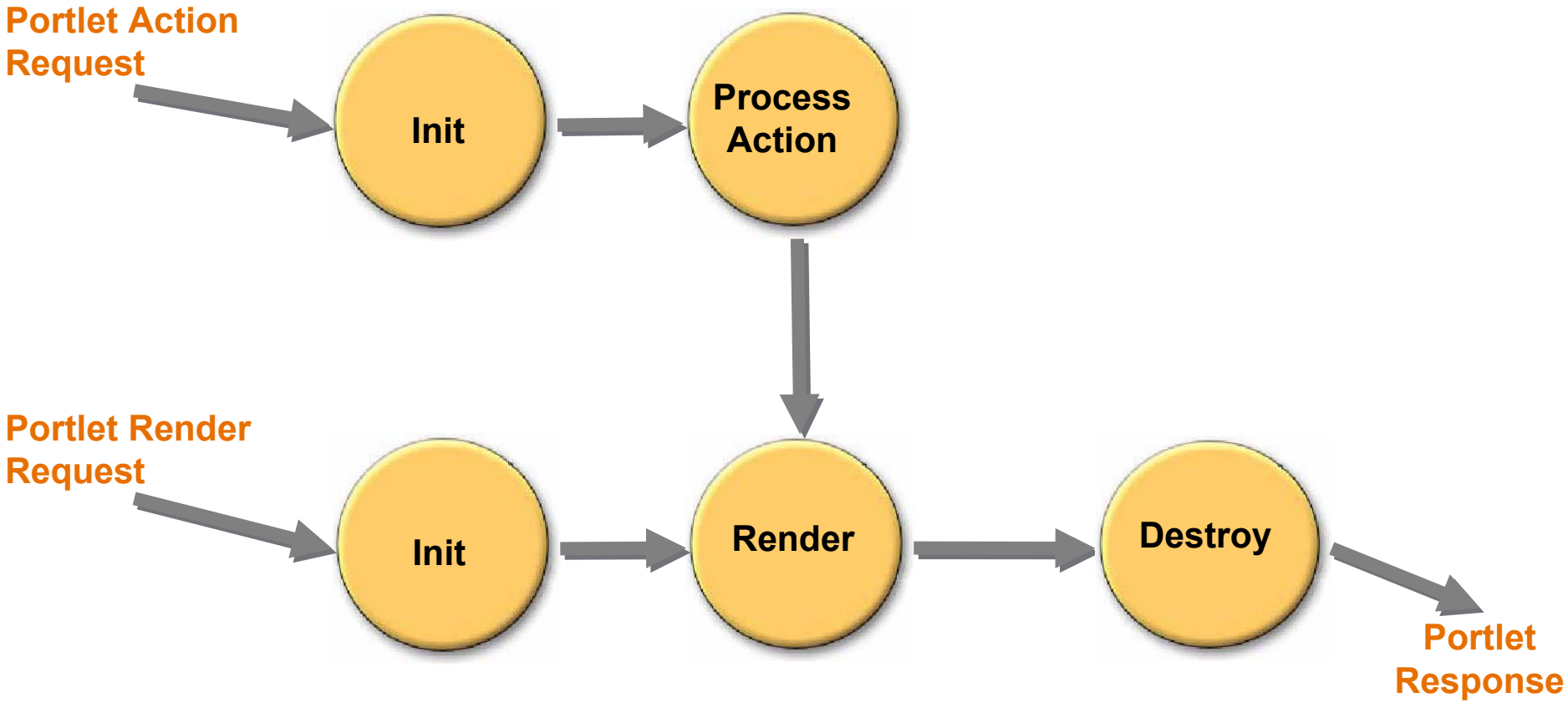
The AJAX lifecycle

Summary and questions

Portlet Application

- User submits a request from a single portlet
- The portlet lifecycle is executed
  - Action
  - Render
- The portlet is updated
  - The entire portal page is sent to the browser

Portlet Lifecycle



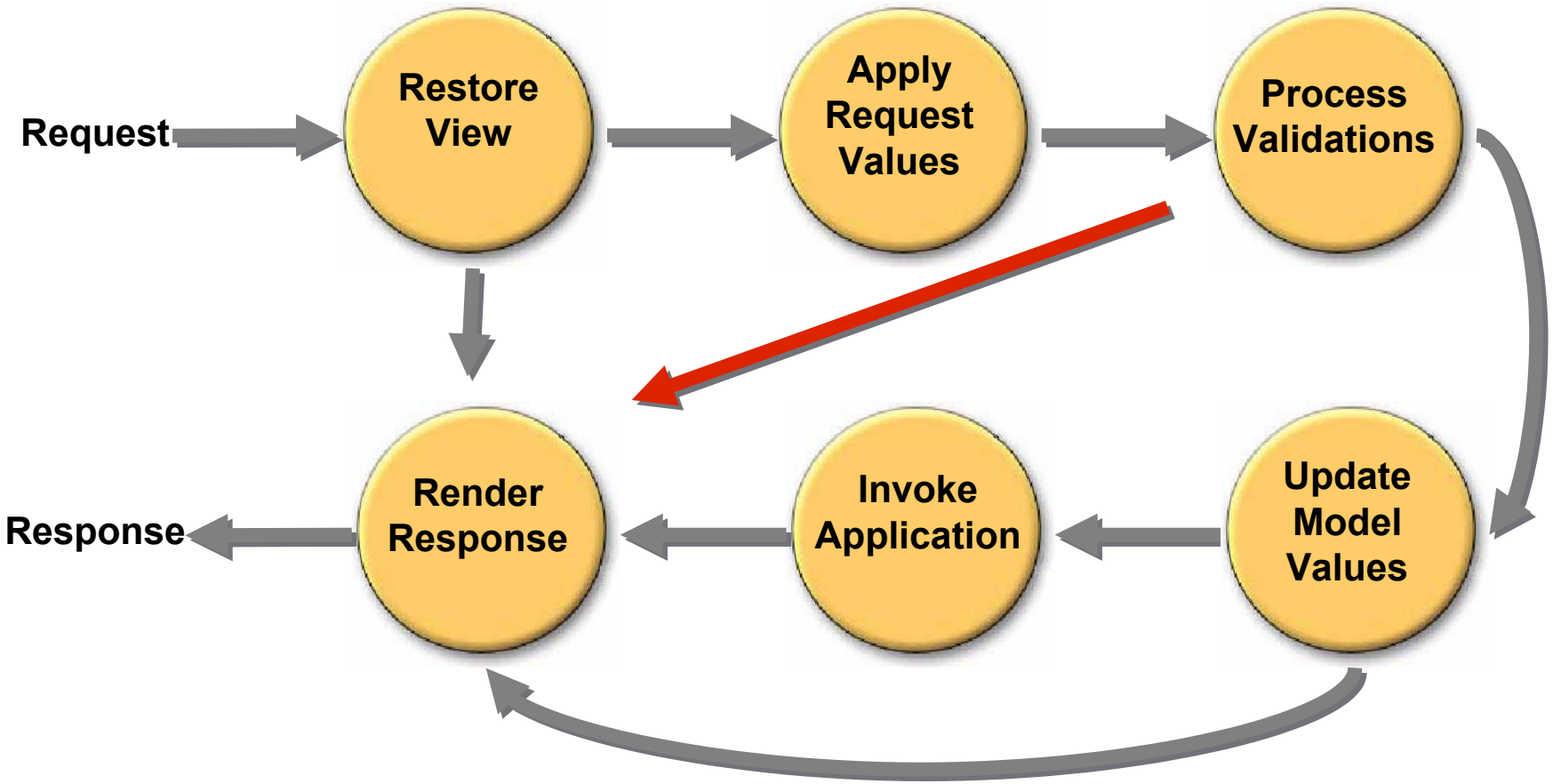
Portlet Issues

- Portlets communicate via the server
- No knowledge of client-side behaviors
- Client-side page contains a number of portlets
- Namespace-encoding prevents dynamic naming and dynamic URL manipulation

Servlet JavaServer Faces Application

- User submits a request
- Lifecycle triggered
  - Well-defined with six phases
- Servlet responds, rendering a single page created with the JavaServer Pages technology (JSP page)

JavaServer Faces Technology Lifecycle



JavaServer Faces Technology Issues

- No direct knowledge of client-side behaviors
  - Components may render JavaScript technology, etc.
- Server-side GUI
- Server-side navigation

AJAX Application

- User triggers a browser action
- Submit occurs programmatically
- Returned data processed by JavaScript technology
- Page updated directly on the client

AJAX Issues

- No direct knowledge of server-side behaviors
  - Assumes the entire application is in the client
- Can break synchronization of client and server

Agenda

Introduction and background

Characteristics of each technology

JavaServer Faces technology and Portlet basics

Mixing them all together

The AJAX lifecycle

Summary and questions

Portlet Basics: Java Technology

- URL encoding

```
ExternalContext context = FacesContext.
      getCurrentInstance().getExternalContext()
;
  …
String encURL = context.encodeActionUrl(url);
String encURL = context.encodeResourceUrl(url);
```

- Name encoding

```
String encName = context.encodeNamespace("foo");
```

Portlet Basics: Java DHTML

- URLs
  Use renderResponse.encodeURL()

```
href='<%= renderResponse.encodeURL(
        renderRequest.getContextPath()
        + "/foo/bar.jsp") %>'
```

- HTML
  - Apply the portlet namespacing tag on IDs

```
<tag name="<portlet:namespace />myTag" … >
```

Basics: DHTML

- JavaScript technology function names
  - Again, apply the portlet namespacing tag

```
function <portlet:namespace />foo() { … }
```

- JavaScript technology events
  - Use EL to retrieve the namespace information

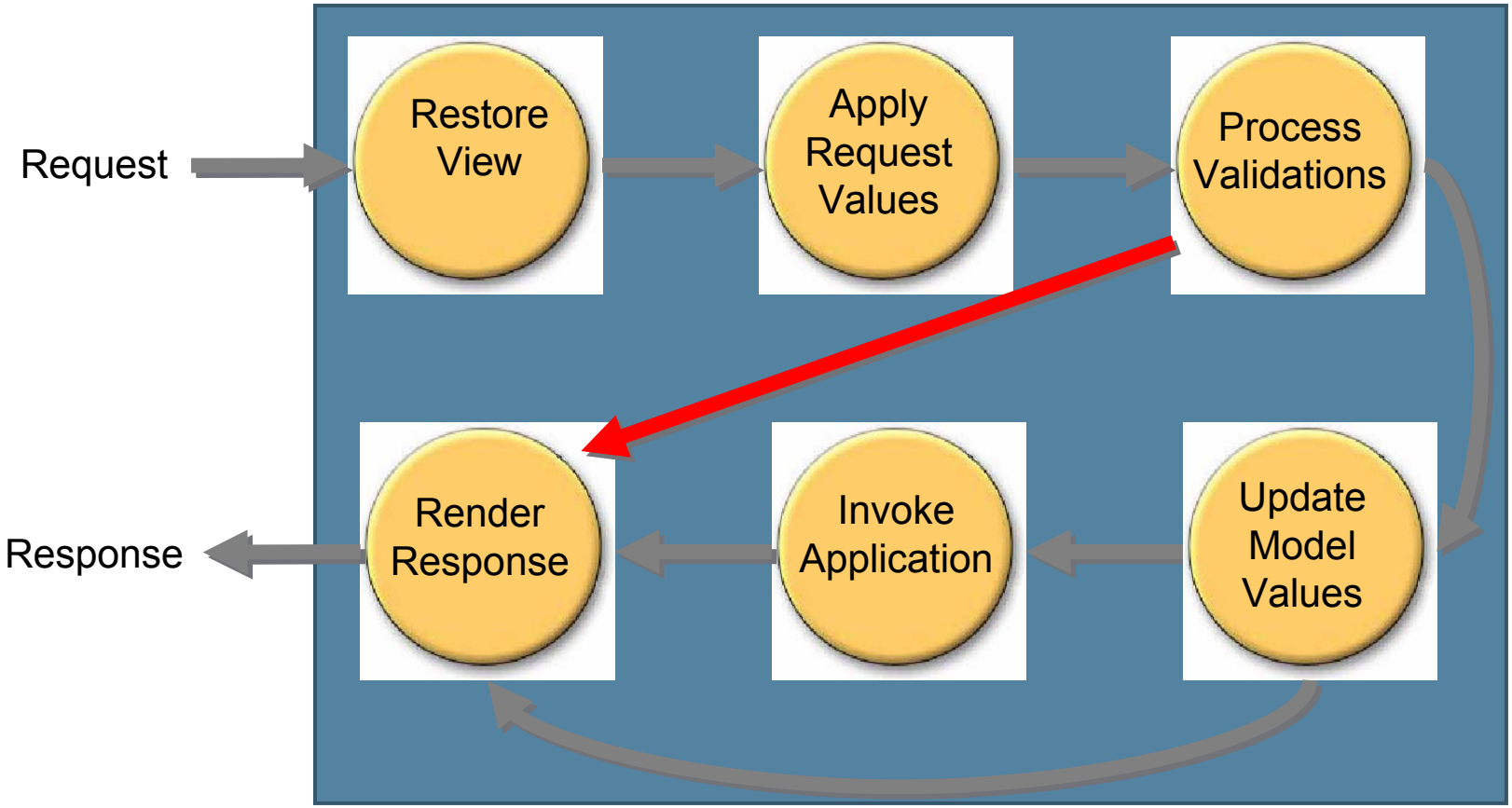
```
<input type="button" onclick=

"#{facesContext.externalContext.response.namespace}
  foo()"
```

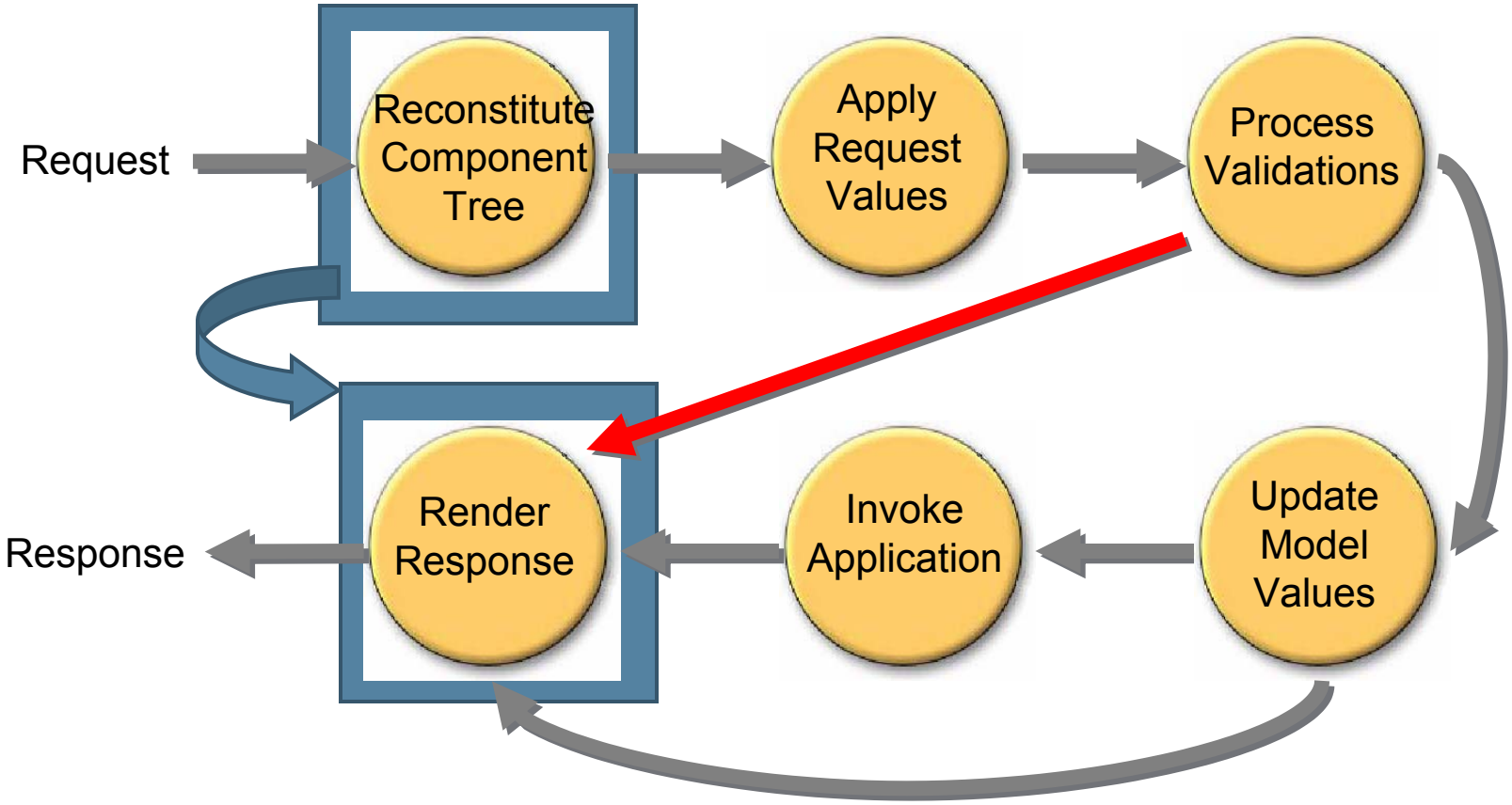
Portlets and JavaServer Faces Technology

- Originally based on FacesGenericPortlet class
  - Mapped the two portlet phases to JavaServer Faces technology six phases
  - Saved parameters in session for render phase
  - Requires session
- JSR 301: Portlet Bridge Specification for JavaServer Faces technology
  - Works transparently
  - Does not assume presence of session
  - Currently multiple versions
    - IBM, Apache, and Sun

Lifecycle Mapping: Action Phase



Lifecycle Mapping: Render Phase



AJAX Sequence of Events

- Trigger a request on the browser portlet page
- Call updating function on server
  - JavaServer Faces technology Lifecycle
  - Data servlet
- Use the response to update the UI
  - Markup returned
    - Update the DOM directly
  - Data returned
    - Convert to DHTML
      - Transform data to required format/markup
      - Insert updated mark-up into page
    - Use data directly
      - Insert or replace data in DOM

Agenda

Introduction and background

Characteristics of each technology

JavaServer Faces technology and Portlet basics

Mixing them all together

The AJAX lifecycle

Summary and questions

Adding AJAX to the Mix

- Request processed by special servlet
  - Return data
    - Pure data stream with no mark-up
  - Return mark-up
    - Pre-build page content
  - Request-response cycle independent of lifecycle
  - Stateless servlet

Adding AJAX to the Mix

- Using JavaServer Faces technology
  - Full JavaServer Faces technology lifecycle
    - Eliminate most of the page's mark-up
    - Pre-formatted mark-up returned to page
  - Partial JavaServer Faces technology lifecycle
    - Interrupted by phase listener
    - Pre-formatted mark-up returned to page

AJAX: Data Servlet

- Generate pure data
  - No need to process data as mark-up
  - Data needs to be formatted for consumption
    - XML, JSON
- Respond
  - Data is processed at the client

AJAX: JavaServer Faces Technology Lifecycle

- Generate page update via lifecycle
  - Run full lifecycle
  - Interrupt lifecycle by special phase listeners
- Filter out unnecessary information
  - Only render required components
  - Remove all other portlets' mark-up
  - Remove themes/skins mark-up
- Respond
  - Send updated minimal mark-up

Agenda

Introduction and background

Characteristics of each technology

JavaServer Faces technology and Portlet basics

Mixing them all together

The AJAX lifecycle

Summary and questions

AJAX Using Full JavaServer Faces Technology Lifecycle

- We need to manage the rendering
  - Only part of the page needs to be updated
  - We don't want any Portal extras
    - No theme information
    - No skins
    - No extra portlet mark-up
- We need to flag that it's an AJAX request
  - Send a special indicator with the request
  - Something must interpret this indicator

Required Artefacts

- AJAXFacesContext
  - AJAXFacesContextFactory
  - AJAXExternalContext
- AJAXRenderkit
  - AJAXRenderkitFactory
  - AJAXRendererWrapper
- Helper classes
  - Utility methods
  - Minimal ResponseWriter

AJAXFacesContext

- Factory
  - Instantiates new AJAXFacesContext
  - Saves original FacesContext object
- AJAXFacesContext
  - Sets special minimal ResponseWriter
- AJAXExternalContext
  - Manages ResponseWriters

AJAXRenderKit

- Factory
  - Wrappers all extant RenderKits to be AJAX-aware
- Renderkit
  - Override getRenderer to wrapper renderers
- AJAXRendererWrapper
  - Manages encodeBegin and encodeEnd

Identifying an AJAX Request

- Use hidden fields on page

```
<input type="hidden" name="MyAJAX" id="MyAJAX" value="">
…
document.getElementById("MyAJAX").value="OK";
```

- Utility method to identify AJAX request

```
public static boolean isAjaxRequest(FacesContext ctx) {
    String ajax = ctx.getExternalContext()
                     .getRequestParameterMap().get("MyAJAX");
    return (null != ajax && "OK".equals(ajax))
}
```

Identify What to Update

- Use hidden fields on page

```
<input type="hidden" name="MyAJAXId" id="MyAJAXId" value="">
<h:panelGrid id="foo" … >
…
document.getElementById("MyAJAXId").value="foo";
```

- Utility method to identify AJAX request

```
public static String getAjaxComponent(FacesContext ctx) {
    return ctx.getExternalContext()
                    .getRequestParameterMap().get("MyAJAXId");
}
```

Client Side

- JavaScript technology
  - Use XmlHttpRequest to call back to server
  - Use DOM manipulations to update page

DEMO

JavaServer Faces Technology, Portlets, and AJAX

Demo Contents

- Show implementation of code
- Create an application that leverages this code

Agenda

Introduction and background

Characteristics of each technology

JavaServer Faces technology and Portlet basics

Mixing them all together

The AJAX lifecycle

Summary and questions

Summary

- JavaServer Faces technology, Portlets, and AJAX work well together
- Creating a generic solution is not too difficult to implement
- The results look cool!

For More Information

- Other sessions
  - TS-9782: Ajax and JavaServer Faces Technology Tooling in Eclipse
  - TS-9511: Ajax with POJC
  - BOF-4664 Dynamic portals
- JSF Central: http://www.jsfcentral.com
- Books
  - Pro JSF and Ajax: Building Rich Internet Components
    - Jonas Jacobi and John Fallows
  - JavaServer Faces: The Complete Reference
    - Ed Burns, Chris Schalk, James Holmes

Q&A

JavaServer™ Faces Technology, AJAX, and Portlets: It's Easy if You Know How!

Brendan Murray

Software Architect
IBM
http://www.ibm.com

TS-6824