



Enterprise Grade Ajax with JavaServer™ Faces

Ed Burns

Senior Staff Engineer

Enterprise Java Platforms



Presentation Goals

- Show that Ajax + JSF is the way to go
- Show one concrete way of using Ajax + JSF
- Learn how to evaluate an Ajax + JSF solution for yourself



Terms



- Enterprise Grade
 - > Robust, Scalable, Well Tested, Industry Proven
 - > Easy to Use
 - > Appropriate to Developer Skillset
 - > Good concept scalability
- Ajax
 - > In a web application, using asynchronous transactions to the server to dynamically update the appearance and behavior of the view shown in the browser.

Agenda



- Why JSF + Ajax?
 - > Web Application Musts
 - > Different approaches
 - > JSF Design Heritage
 - > JSF Features for Ajax
 - > Problems and Solutions
- One Solution:
Project Dynamic Faces
- Future Directions
- Summary

All Web Applications Must Have...



All Web Applications Must Have...

- Data Conversion and Validation
- Page Flow
- Database integration



- I18N, L10N, A11Y
- Support CSS, Markup based layout
- User Friendliness!

Approaches for using JSF + Ajax

In Order of Decreasing Complexity

- “Naked” Ajax (Frank Zametti)
 - > You'll gain a deep understanding of Ajax techniques
 - > Have to handle all XMLHttpRequest interactions yourself. (SetTimeout anyone?)
 - > Have to handle cross browser quirks (Legendary pain)
 - > You'll end up writing a framework yourself anyway!
- Use a JavaScript Framework (Dojo, DWR, etc)
 - > Many available, some really good
 - > Web app “Musts” provided by JSF still have to be integrated with the framework.

Approaches for using JSF + Ajax

In Order of Decreasing Complexity

- JSF + DynaFaces

- > No JavaScript knowledge required
- > Can do more powerful stuff if you like JavaScript
- > Only JSF + Ajax solution that correctly handles context sensitive Ajax component interaction.



- > Currently only implemented with JSF 1.2
- > Still in early access: needs more testing

- Using Ajax enabled JSF Components

- > Minimal Ajax awareness required
- > Some nice components available, even for JSF 1.1
- > If you can't find one, you'll have to write it yourself, best use DynaFaces



The Big Question

Is Ajax different enough from common web application UI practices that it warrants a completely different approach?

No.

Lots of people will still sell you stuff to learn Ajax, though!

One Possible Answer

JSF + Ajax

- Use JSF for Web App “Musts”
- Provide some extra usage contract to enable Ajax support
- Several options currently
 - > DynaFaces
 - > IceFaces
 - > Ajax4JSF
 - > AjaxAnywhere
 - > Backbase

Why JSF + AJAX?

- OO Design of JSF was ready for AJAX when AJAX wasn't cool.
- Key Features of JSF that make it AJAX friendly
 - > Flexible and extensible component model
 - > Well defined Request Processing Lifecycle
 - > Flexible and extensible rendering model
- Concepts that enable AJAX
 - > Encapsulation: ability to hide JavaScript from the page author, but show it to the component author
 - > State Management: easily keep client and server state in synch

Typical Ajax Problems Solved by Some JSF + Ajax Solutions

- Cross Browser Differences: wrap a JavaScript Framework that abstracts them.
- I18N, L10N: JSF Support Just works, even for Ajax
- Script management and versioning: Apache Shale
- “too chatty” XmlHttpRequest usage: batch events
- Using XML for transport requires extra finesse for some common HTML elements, such as ` `;
- Web app “Musts” handled naturally by JSF

The Best JSF + Ajax Solution

Project Dynamic Faces

- Brings the power of AJAX to existing and future JSF developed applications.
- Based on “Avatar” idea from Jacob Hookom, creator of Facelets, and shown at JavaOne 2006
- Small footprint developer usage contract
- Big Increase in Ajax capability

Demonstrations

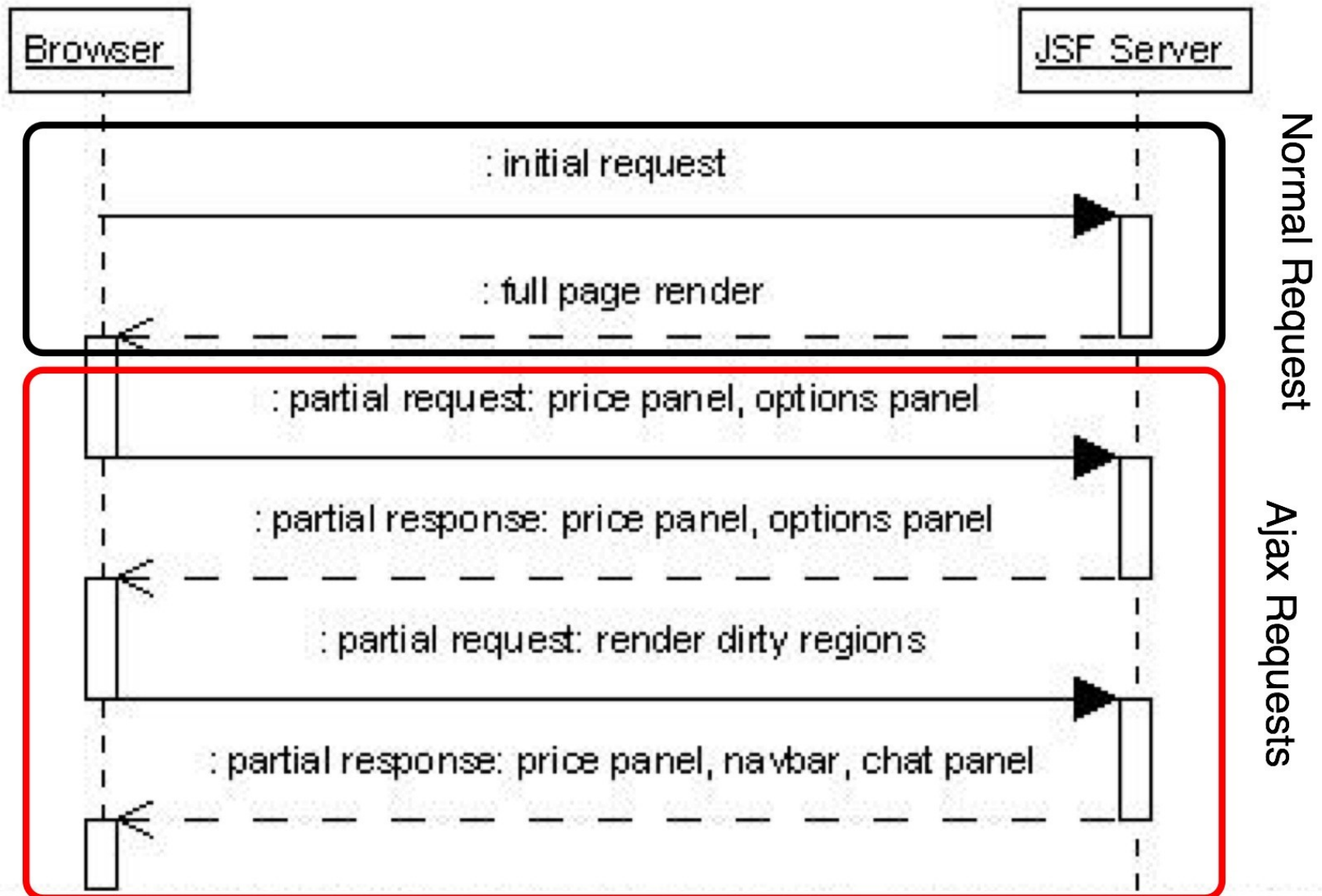


Project Dynamic Faces

The Basic Idea

- Expose the JSF Lifecycle to the browser via Ajax
 - > Allow operating on the entire view, or a part of it
 - > Allow controlling the Lifecycle via Ajax
 - > Allow the server to change appearance and behavior of current page

The Importance of the Lifecycle



Views and Partial Views


Sun Java(TM) ...

CarStore

CarStore

An overview ...

Using DynaFa...



Duke's Vulgar Luxury Car

Those Joneses next door have everything, don't they? Well, show them that you're the one pulling in the big bucks by driving up in this loaded chassis.

Base Price 20700
Your Price 22650

Buy

Options and Packages

Custom

Standard

performance

Deluxe

Engine V8

Brakes ☐ disc ☒ drum

Suspension performance

Speakers ☐ 4 ☒ 6

Audio ☐ Standard ☒ premium

Transmission manual

Other Options

☒ Sunroof
 ☒ Cruise Control
 ☒ Keyless Entry

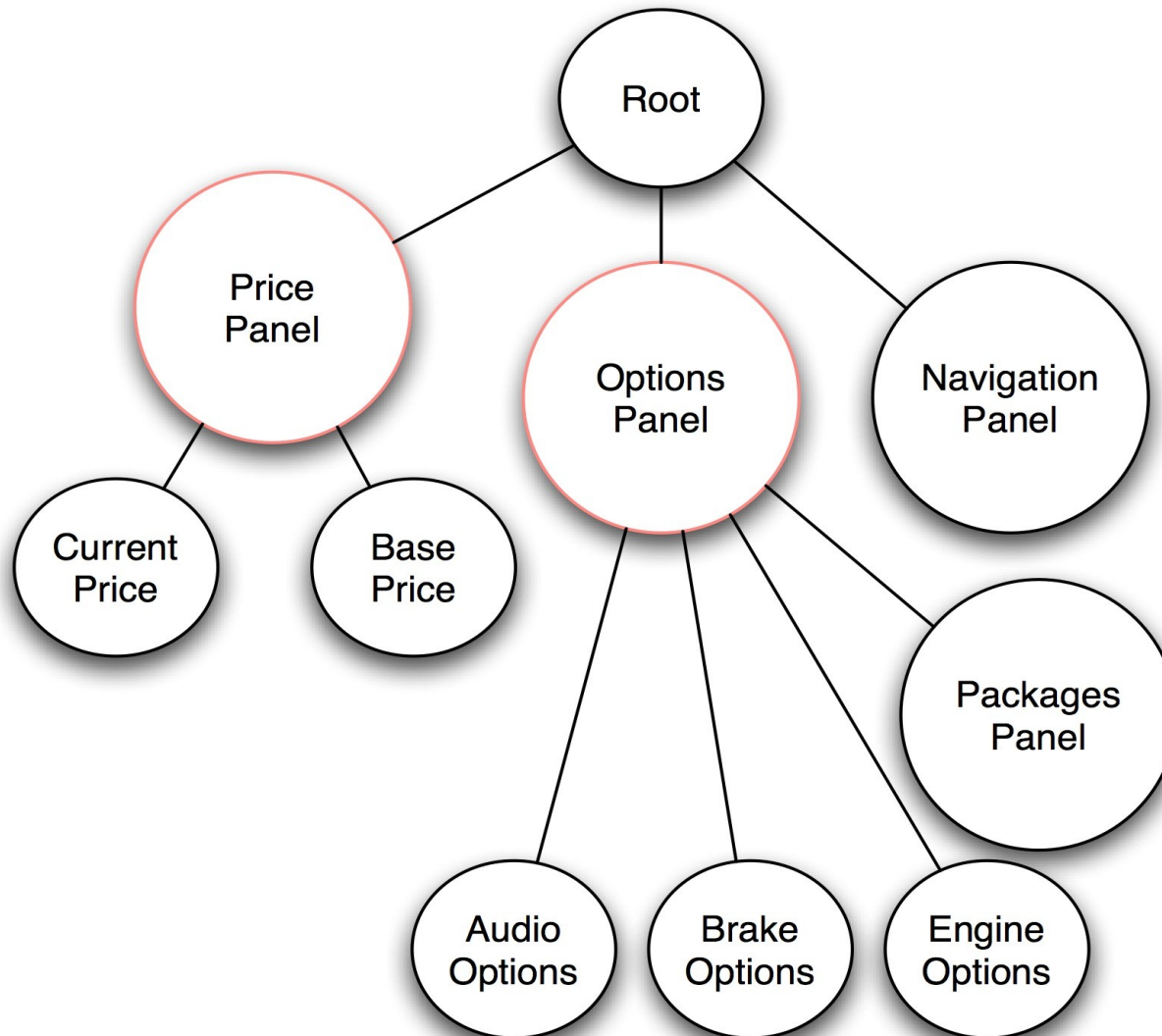
☐ Security System
 ☐ Ski Rack
 ☒ Tow Package

☐ GPS

Recalculate Buy

[Documentation for this demo](#)

Views and Partial Views



DynaFaces — Usage Patterns

- Page Author

- > Use AJAX enabled components
- > Use AjaxZone tag to AJAXify regions of the page
- > Use provided JavaScript library to AJAXify page elements and components

Increasing Complexity
↓

- Component Author

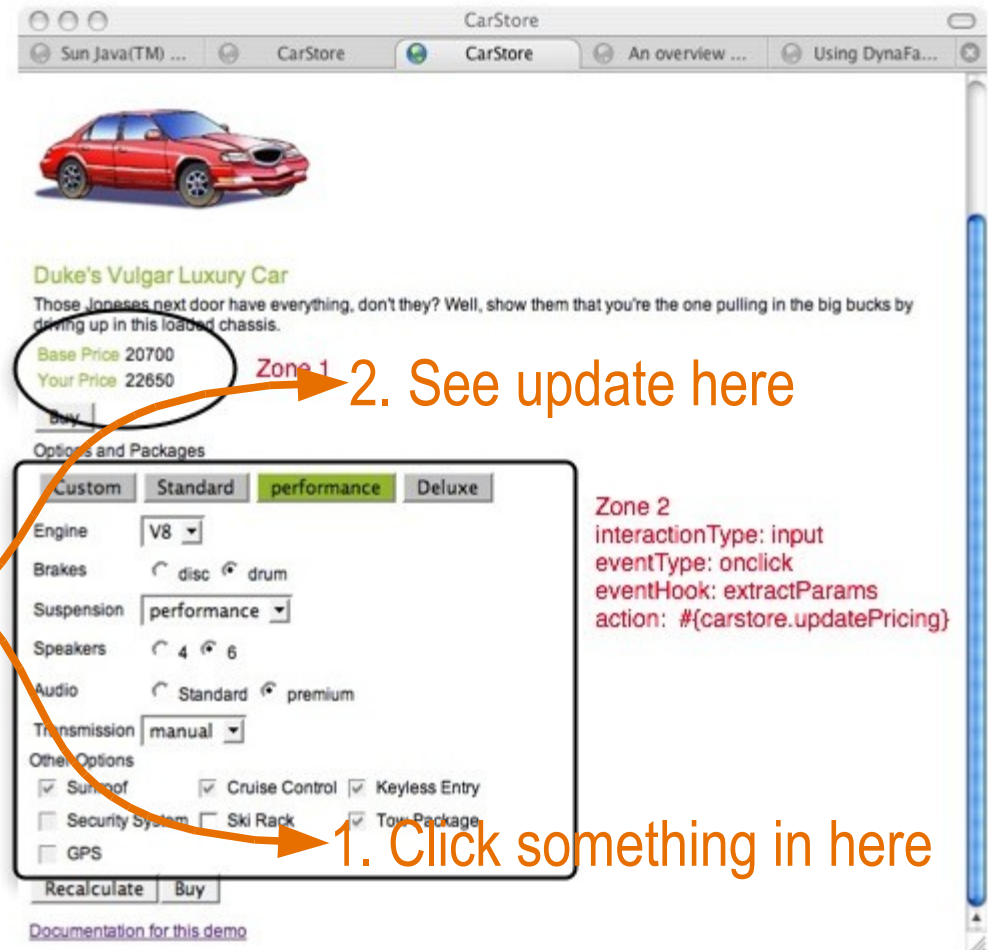
- > Use provided JavaScript library in custom components
- > Write your own JavaScript that talks directly to the HTTP protocol and the XML application defined by DynaFaces

Increasing Complexity
↓

DynaFaces Usage Patterns

Using AjaxZones

- The easiest way to AJAXify an existing application
- Demarcate one or more AJAX zones within a page
- Zones will refresh via AJAX, without full page refresh.
- Action in one zone causes reaction in another zone



Zone 1

Base Price 20700
Your Price 22650

Zone 2

interactionType: input
eventType: onclick
eventHook: extractParams
action: #{carstore.updatePricing}

1. Click something in here

2. See update here

Demonstration



DynaFaces Usage Patterns

Using AjaxZones

```
<jsfExt:ajaxZone id="zone1"> <h:panelGrid columns="2">
```

```
Base Price <h:outputText binding="#{currentModel.basePrice}" />
```

```
Your Price <h:outputText value="#{currentModel.currentPrice}" />  
</h:panelGrid> </jsfExt:ajaxZone>
```

```
<jsfExt:ajaxZone id="zone2"  
  action="#{carstore.currentModel.updatePricing}">
```

```
Option Packages <h:panelGrid columns="4">
```

```
  <h:commandButton value="Custom"  
    actionListener="#{carstore.choosePackage}" />
```

```
  <h:commandButton  
    value="Standard" actionListener="#{carstore.choosePackage}" />
```

```
</h:panelGrid> <h:panelGrid columns="2">
```

```
Engine <h:selectOneMenu binding="#{currentModel.components.engine}" />
```

```
Breaks <h:selectOneRadio binding="#{currentModel.components.brake}" />
```

```
Suspension
```

```
  <h:selectOneMenu binding="#{currentModel.components.suspension}" />
```

```
Speakers <h:selectOneRadio
```

```
  binding="#{currentModel.components.speaker}" />
```

```
</jsfExt:ajaxZone>
```

DynaFaces Usage Patterns

Using AjaxZones — available attributes

- `action` (optional)
 - > MethodExpression to invoke when the request processing lifecycle in which this zone is being processed reaches its `invokeApplication` phase.
- `immediate` (optional): Just like `commandButton`
- `inspectElement`: (optional)
 - > User defined JavaScript function that takes an HTML element and returns `true` or `false` depending on whether or not this element should be AJAXified. The default `inspectElement` function will return `true` for every child of this zone that is an HTML `input`, `button`, or `option` tag.
- `eventType`: (optional) JavaScript event to cause the AJAX request. If not specified, defaults to “click”.

DynaFaces Usage Patterns

Using AjaxZones — available attributes

- `collectPostData` (optional)
 - > User defined JavaScript function called when the `eventType` event occurs. Extracts name=value pairs to send in AJAX request. If not specified, the following name=value pairs are sent
 - > The name=value pair of any input field within the zone.
 - > For any radio button or menu, the name=value pair of the currently selected choice.
 - > If the activated component that resulted in this callback being called is a button, the name=value pair for that button only. Any other buttons within the zone do not have their name=value pairs contributed to the AJAX request.
- `postReplace` (optional)
 - > This optional attribute names a function to be called after the new content from the server for this zone has been installed into the view. Used for jMaki support.

DynaFaces Usage Patterns

Using AjaxZones — available attributes

- `replaceElement` (optional)
 - > This optional attribute is or names a JavaScript function that will be called when the system needs to replace a chunk of markup in the view based on the return from the server. The default implementation is sufficient for most cases.
- `getCallbackData` (optional)
 - > This optional attribute names a function to be called to provide a closure argument that will be passed to the ajax request and made available to the ajax response in the `replaceElement` or `postReplace` functions.

DynaFaces Usage Patterns

Using DynaFaces.fireAjaxTransaction

- Defined in built-in JavaScript library.
- When called, causes an AJAX transaction to the Faces server.
- Many options for customizing the transaction.

Order Total:\$452.16				
1	BX Latex Surgical Gloves 3M			\$10.40
22	BX 40cc Syringe	Flownder Medical		\$441.76

Id	Description	UOM	Qty	
59339	40cc Syringe	BX	22	Add Item
45439	Latex Surgical Gloves	BX	1	Add Item
46787	Bed Restraint	DZ		Add Item
54333	Small Cane Tip	EA		Add Item
78799	Large Cane Tip	EA		Add Item

Demonstration



DynaFaces Usage Patterns

Using DynaFaces.fireAjaxTransaction

```
<h:commandButton value="Add Item"
  action="#{orderEntry.addProduct}"
  onclick="new
DynaFaces.fireAjaxTransaction(this);" />
```

- Useful when you want the AJAX transaction to happen as soon as the script is executed.
- Default action just does a refresh of the whole view via AJAX, using JavaScript DOM methods to update the elements.
- Can add options to choose exactly which subtrees get sent, processed, and re-rendered.

DynaFaces Usage Patterns

DynaFaces.fireAjaxTransaction General Form

- Generally used in a tag attribute

```
<ANY_HTML_OR_JSF_ELEMENT  
    on|EVENT|="DynaFaces.fireAjaxTransaction(this,{ |  
    OPTIONS| } ) ;" />
```

- Where
 - > ANY_HTML_OR_JSF_ELEMENT is any HTML element or JSF tag
 - > on|EVENT| is any JavaScript event type, such as onclick
 - > { |OPTIONS| } is an optional argument. If present, it is a JavaScript associative array containing any options desired for this transaction.
- DynaFaces.fireAjaxTransaction() may be used from non event-handler JavaScript as well.

DynaFaces Usage Patterns

Using `DynaFaces.fireAjaxTransaction`

- `execute` (optional)
 - > Comma separated list of client ids to be traversed on the “execute” portion of the JSF Request Processing Lifecycle (everything but render). If not specified, the value of the render option is used. “none” indicates that the view must not be traversed during the execute portion of the lifecycle.
- `render` (optional)
 - > Comma separated list of client ids to be traversed on the “render” portion of the JSF Request Processing Lifecycle. If not specified it's up to the server to decide what to re-render. By default the whole view is re-rendered. “none” indicates that the view must not be rendered.
- `inputs` (optional)
 - > Comma separated list of clientIds for which the value should be sent in the AJAX request. If not specified, all input components in the current form are submitted.

DynaFaces Usage Patterns

Using `DynaFaces.fireAjaxTransaction` — options

`postReplace` (optional)

User defined JavaScript function called after element replacement. Useful when integrating with jMaki. If not specified, any scripts present in the markup to be rendered are evaluated.

`replaceElement` (optional)

User defined JavaScript function called for element replacement. If not specified, default element replacement occurs.

`getCallbackData` (optional)

User defined JavaScript function, returns closure object that is passed to the `replaceElement` or `postReplace` functions.

`immediate` (optional)

boolean value that tells the server to set the immediate option, for this transaction only, on any input or command components in the traversal.

DynaFaces Usage Patterns

Using `DynaFaces.fireAjaxTransaction` — options

`asynchronous` (optional)

Should this transaction be asynchronous (the default) or synchronous

`xjson` (optional)

Any JSON data that should be sent along with the transaction, as the value of the X-JSON header

`closure` (optional)

Closure object that is passed to the `replaceElement` or `postReplace` functions.

`immediate` (optional)

boolean value that tells the server to set the immediate option, for this transaction only, on any input or command components in the traversal.

DynaFaces Usage Patterns

Dispatching JSF Events via Ajax

ActionEvent Example

Submit queued actionEvents.

submit via ajax

← 2. press this button once.

ActionListener Output:

[ajax] [ajax] [ajax]

← 3. See these appear via AJAX.

Queue additional ActionEvents.

queue action

← 1. Press this button 3 times.

- Allow queuing of arbitrary `javax.faces.event.FacesEvent` subclasses from JavaScript directly into JSF Lifecycle.
- JavaScript classes for standard `ValueChangeEvent` and `ActionEvent` classes.
- Use JavaScript “subclassing” for custom events.

Demonstration



DynaFaces Usage Patterns

Dispatching JSF Events via Ajax ActionEvent

```
<script type='text/javascript'>
    function queueEvent() {
        var actionEvent = new DynaFaces.ActionEvent("ajax",
            DynaFaces.PhaseId.INVOKE_APPLICATION);
        DynaFaces.queueFacesEvent(actionEvent);
        return false;
    }
</script>

<h:commandButton id="ajax" value="submit via ajax"
    onclick="DynaFaces.fireAjaxTransaction(this, { render: 'label' });
    return false;" actionListener="#{bean.processAction}" />

<h:outputText id="label" value="#{requestScope.actionEvents}" />

<input type="submit" name="queueAction" id="queueAction"
    value="queue action" onclick="queueEvent(); return false;" />
```

- Non-JSF button used to queue action events.
- JSF commandButton used to fire the AJAX transaction. Requests re-render of the label component.
- actionListener updates request scoped property actionEvents, which is output from the label component.

DynaFaces Usage Patterns

Dispatching JSF Events via Ajax-ValueChangeEvent

```
<script type='text/javascript'>
    function queueEvent() {
        var valueChangeEvent = new DynaFaces.ValueChangeEvent("input",
            DynaFaces.PhaseId.UPDATE_MODEL_VALUES,
            "oldValue", "newValue");
        DynaFaces.queueFacesEvent(valueChangeEvent);
        return false;
    }
</script>

<h:inputText id="input" valueChangeListener="#{bean.valueChange}" />

<h:commandButton value="submit via ajax"
    onclick="DynaFaces.fireAjaxTransaction(this, { execute: 'input',
    render: 'label,input', inputs: 'input' }); return false;" />

<h:outputText id="label" value="#{requestScope.valueChangeEvents}" />

<input type="submit" name="newValue" id="newValue" value="queue event"
    onclick="queueEvent(); return false;" />
```

- Non-JSF button used to queue value change events.
- JSF commandButton to fire the AJAX transaction. Requests execution of input component, and re-render of the label component. The inputs attribute is used to cause only the input component to be submitted.
- valueChangeListener updates request scoped property valueChangeEvents, which is output from the label component.

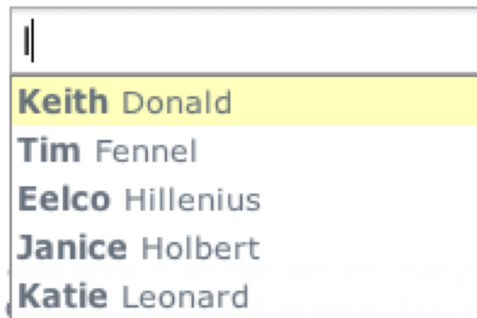
DynaFaces Usage Patterns

Dispatching JSF Events via Ajax-Custom Event Class

Auto Suggest

The token AJAX widget for frameworks. This example shows how a developer can easily customize the presentation of the widget, just as they would with any other content on the page with server-side templating.

1. Type into this field.
2. 'onkeypress' event fires.
3. Custom event is instantiated, queued, and a DynaFaces Ajax Transaction is initiated
4. 'onComplete' is called, and text updates.



Source 

Contextual Auto Suggest

Ideally, we want everything to work just as we are used to with standard JSF, referencing/using variables that are contextual. This isn't an issue for JavaServer Faces.

...times you can run into issues with the page. The example below shows that this

- Uses JavaScript “subclassing” for custom events.
- Combined with a custom component `<e:serverSuggest />`
- This component overrides the standard `UIComponent.broadcast(FacesEvent event)` to look for an instance of `SuggestEvent`, from which it extracts the submitted information and dispatched to the Renderer.

Important Links

- Project Glassfish
<http://glassfish.dev.java.net>
- JSF-Extensions, DynaFaces
<http://jsf-extensions.dev.java.net>
Live Demos, Screencasts, These Slides!
- Project jMaki
<http://ajax.dev.java.net>
- Sun Blueprints Ajax Components
<http://java.sun.com/blueprints/ajax.html>
Live Demos at
<http://sunapp2.why.com/AjaxSamples2/>

Future Directions

- DynaFaces project is open source (CDDL)
- Informal collaboration agreement with Exadel (Ajax4JSF), ICESoft (ICEFaces)
- Work will be harvested for JSF 2.0 Standard
- File JSF 2.0 JSR before end of the year



Summary

- JSF + Ajax makes perfect sense
- Questions to ask yourself about JSF + Ajax solutions
 - > How steep is the learning curve?
 - > Is JavaScript knowledge required for productive use?
 - > How easy is partial page update?
 - > Is JavaScript accessible to advanced users?
- Naturally, DynaFaces has good answers for these!



JSF + Phobos

Ed Burns

ed.burns@sun.com

