JSF 2 and beyond: Keeping progress coming

Andy Schwartz - Oracle Corporation

Dan Allen & Pete Muir - Red Hat, Inc.

Goals

See how far JSF 2 has come,

explore the community's role and

take a glimpse at JSF 2.next

Join in!

Twitter hashtag: #jsf2nex

Join in!

JSF 2 and beyond: BOF

Tonight @ 20:00 in Room 2!

Andy Schwartz

http://andyschwartz.wordpress.com

- Software engineer at Oracle Corporation
- Architect on ADF Faces project team
- Member of the JSR-314 (JSF 2) Expert Group

Pete Muir

http://in.relation.to/Bloggers/Pete

- Principal Software Engineer at Red Hat, Inc
- Seam and Weld (JSR-299 RI) project lead
- Member of the JSR-314 (JSF 2) Expert Group

Dan Allen

http://mojavelinux.com

- Senior Software Engineer at Red Hat, Inc
- Author of Seam in Action
- Seam and Weld project member
- Member of the JSR-314 (JSF 2) Expert Group

Many faces of JSF 2

pokom

eal

ci

ke

ekt

eh

Byrne

elle

n King

Imre Oswald

h i a s W e s

x a n d r S m

m a n u e l B e

Topic areas

- View (Andy)
  - Facelets and VDL
  - Ajax & behaviors
  - Components & state saving
- Controller (Dan)
  - “Bookmarkability”
  - Navigation
  - Resource loading
- Model (Pete)
  - Components and EL
  - Validation
  - Error handling
- Pain relief
- Community (Dan)

View declaration

Facelets, View Declaration Language API

The problem

JSP pain points

- Content vs component tree creation
- Grunge
  - Tag class
  - Tag library
- Mixing presentation with logic
- Translation/compilation
- Stateful tags

The solution

Facelets

(Thanks, Jacob!)

Breaking free with Facelets

- View definition optimized for JSF
- XHTML + tags (no scriptlets)
- Default, stateless tag handling
- Simplified tag library configuration
- No more translation/compilation
- Templating

The problem revisited

But, Facelets isn't standard :(

The solution revisited

Now it is!

The solution 2.0

- JSF 2.0 includes Facelets in the spec
- Same features, some enhancements
- Facelets is now preferred over JSP
  - Most new functionality not available in JSP
- Also new: View Declaration Language APIs

View Declaration Language API

- Common infrastructure for VDLs
- Encapsulates tree building, state saving
- Encourage innovation in VDL space
  - JSF Templating
  - Gracelets
  - Any other ideas?

Facelets and VDL: JSF2.next

- Facelets XHTML vs. XML
- XSD for Facelets
- Facelets/JSP compatibility
- Whitespace handling
- Are Facelets APIs complete?
- Are VDL APIs complete?

Component development

Java components, composite components

The problem

Component development is hard!

The problem in detail

- Too many artifacts
  - UIComponent class
  - Renderer class
  - Tag class
  - tld
  - lots of faces-config.xml
- Ouch!

The solution: Take 1

Simplify Java component development

The solution: Take 1

- Annotations replace `faces-config.xml`
- Default handlers replace tag classes
- Facelets `taglib.xml` replaces tld grunge
- Simplified state saving
  – More on this in a bit...
- Better, but good enough?

The solution: Take 2

Composite components!

Composite components

- Easy component creation (via Facelets)
  - It's not just for JSF gurus any more
- Defined using a single Facelets file
- No external configuration
- Conventions define tag namespace/name
- No Java code required

Composite component definition

- `<composite:interface>`
  – defines tool/runtime metadata
- `<composite:implementation>`
  – defines content and behavior
- Composite tags for inserting children
- Attribute access via `#{cc.attrs}`
- Client id access `#{cc.clientId}`

Composite component definition

resources/foo/greeting.xhtml

```
<composite:interface>
   <composite:attribute name="name" default="World"/>
</composite:interface>

<composite:implementation>
   Hello, #{cc.attrs.name}!
</composite:implementation>
```

Composite component usage

```
<html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:foo="http://java.sun.com/jsf/composite/foo">
   <body>
      <foo:greeting name="Devoxx"/>
   </body>
</html>
```

Composite components

- Definitions live in web root or JAR
- Optional Java/Groovy backing file
- Optional `.properties` file
- Optional supporting resources
- Attach listeners, converters, validators, behaviors

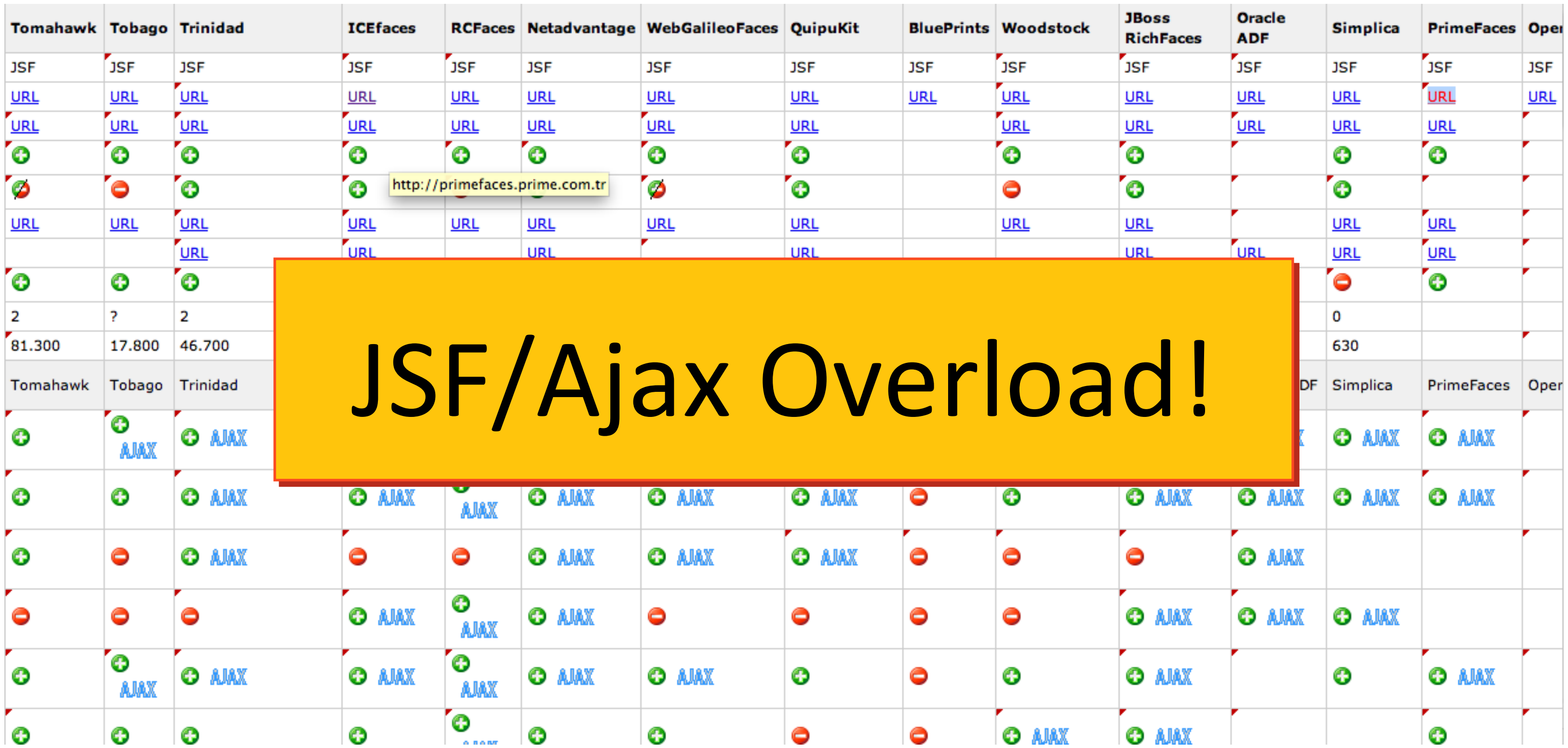
Component development: JSF2.next

- Possible to simplify further?
- Hybrid tag libraries (composites + Java)
- Resource location (`WEB-INF/resources`)
- Java/Groovy backing class naming
- Insert vs. render children

Ajax

jsf.ajax.request(), <f:ajax>, Ajax Java APIs, and tree visiting

The problem



Tomahawk	Tobago	Trinidad	ICEfaces	RCFaces	Netadvantage	WebGalileoFaces	QuipuKit	BluePrints	Woodstock	JBoss RichFaces	Oracle ADF	Simplica	PrimeFaces	Oper
JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF	JSF
URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL	URL
URL	URL	URL	URL	URL	URL	URL	URL		URL	URL	URL	URL	URL	
⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕		⊕	⊕		⊕	⊕	
⊘	⊖	⊕	⊕			⊘	⊕		⊖	⊕		⊕		
URL	URL	URL	URL	URL	URL	URL	URL		URL	URL		URL	URL	
		URL	URL		URL		URL			URL	URL	URL	URL	
⊕	⊕	⊕										⊖	⊕	
2	?	2										0		
81.300	17.800	46.700										630		
Tomahawk	Tobago	Trinidad									DF	Simplica	PrimeFaces	Oper
⊕	⊕ AJAX	⊕ AJAX										⊕ AJAX	⊕ AJAX	
⊕	⊕	⊕ AJAX	⊕ AJAX	AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊖	⊕	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	
⊕	⊖	⊕ AJAX	⊖	⊖	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊖	⊖	⊖	⊕ AJAX			
⊖	⊖	⊖	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊖	⊖	⊖	⊖	⊕ AJAX	⊕ AJAX	⊕ AJAX		
⊕	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕ AJAX	⊕	⊖	⊕	⊕ AJAX		⊕	⊕ AJAX	
⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊖	⊖	⊕ AJAX	⊕ AJAX			⊕	

http://primefaces.prime.com.tr

JSF/Ajax Overload!

Where things went wrong

- Everyone has a solution
- No two solutions are compatible
- Sad application developers

The solution

Standard Ajax APIs

The solution in detail

- Start with a programmatic API
- `jsf.ajax.request()`
- Add in some declarative support
- `<f:ajax>`
- Don't forget about the server side
  - PartialViewContext
  - PartialResponseWriter

jsf.ajax.request()

- Java EE's first JavaScript API!
- Performs a partial page update
- Caller specifies execute/render ids
  - Or keywords: @all, @form, @this, @none
- `jsf.ajax.request()` takes care of the rest
- Supports notifications of events/errors

jsf.ajax.request()

```
<h:outputScript name="jsf.js" library="javax.faces"/>
…
<h:commandButton value="Do something Ajaxy"
   onclick="jsf.ajax.request(this, event, {render: 'out'}); return false;"/>
…
<h:outputText id="out" value="Update me!"/>
```

<f:ajax>

- Declarative mapping for `jsf.ajax.request()`
- Attach via nesting or wrapping

<f:ajax> nesting

```
<h:commandButton value="Do something Ajaxy">
   <f:ajax render="out"/>
</h:commandButton>
...
<h:outputText id="out" value="Update me!"/>
```

<f:ajax> wrapping

```
<f:ajax render="out"/>
  <h:commandButton value="Do something Ajaxy"/>
  <h:commandButton value="Do something else"/>
  <h:commandButton value="One more here"/>
</f:ajax>
...
<h:outputText id="out" value="Update me!"/>
```

<f:ajax> client events

```
<h:commandButton>
  <f:ajax event="mouseover"/>
</h:commandButton>
...
<h:inputText>
  <f:ajax event="focus"/>
</h:commandButton>
```

Ajax Java APIs

- AjaxBehavior
- PartialViewContext
- Read/write access to execute/render lists
- `processPartial()`
- PartialResponseWriter
- New tree visitor API

Ajax: JSF2.next

- Ajax debugging
- Fallback
- Id round-tripping
- Out-of-band/GET requests
- Event collapsing
- File upload

Behaviors

ClientBehavior, ClientBehaviorHolder

The problem

It’s not just about Ajax

Think bigger

- Avoid tight coupling
- Allow arbitrary behaviors
- Allow arbitrary components to participate

The solution

New contract:
separate behavior from component

ClientBehavior API

- New type of attached object
  - Like converter, validator
- Attached to component by “event”
- Contributes scripts to markup
- Also can participate in decode

ClientBehavior sample

```
@FacesBehavior("org.demo.behavior.Greet")
public class GreetBehavior extends ClientBehaviorBase {

  @Override
  public String getScript(ClientBehaviorContext ctx) {
    return "alert('Hello, World!')";
  }
}
```

ClientBehavior sample

```
<h:commandButton value="Do something Ajaxy">
   <f:ajax/>
</h:commandButton>

<h:commandButton value="Say Hello">
   <foo:greet/>
</h:commandButton>
```

What else is possible?

- Client-side validation
- DOM manipulation
- Tooltips, hover content
- Logging
- Confirmation
- Key handling

ClientBehaviorHolder API

- Contract by which behaviors are attached
- Remember `EditableValueHolder`?
- `addClientBehavior(eventName, behavior)`
- Specifies component-specific events
- Specifies optional default event

ClientBehaviorHolder API

- UIComponentBase has base support
- Implemented by all standard components
- Yours can too!
- Renderers responsible for retrieving and rendering ClientBehavior scripts

Behaviors: JSF2.next

- Other standard client behaviors?
- Other categories of behaviors?
  - Phase behavior
- Pre-decode behavior execution
- Rendering utilities

State saving

Partial state saving, state helper

The problem

State saving is nasty

State saving lunacy

```
public Object saveState(FacesContext ctx) {
   if (_values == null) {
      _values = new Object[10];
   }
   _values[0] = super.saveState(ctx);
   _values[1] = accesskey;
   _values[2] = alt;
   _values[3] = dir;
   _values[4] = disabled;
   _values[5] = image;
   _values[6] = label;
   _values[7] = lang;
   _values[8] = onblur;
   _values[9] = onchange;
   return _values;
}
```

```
public void restoreState(
      FacesContext ctx, Object _state) {
   _values = (Object[]) state;
   super.restoreState(ctx, _values[0]);
   this.accesskey = (java.lang.String) _values[1];
   this.alt = (java.lang.String) _values[2];
   this.dir = (java.lang.String) _values[3];
   this.disabled = (java.lang.Boolean) _values[4];
   this.image = (java.lang.String) _values[5];
   this.label = (java.lang.String) _values[6];
   this.lang = (java.lang.String) _values[7];
   this.onblur = (java.lang.String) _values[8];
   this.onchange = (java.lang.String) _values[9];
}
```

Another problem

State saving is expensive

State overhead

- State saving == component developer tax
  – Do I really need to implement saveState and restoreState?
- Full component tree state not small
  – Where do you want it? Session? Client?

The solution

Partial state saving for smaller state.

State helper utilites for happier component developers.

Partial state saving

- Why save the full component tree?
- Initial component tree is accessible
  - Just need to re-execute the tags
- Initial component tree isn't sufficient
- Also need any state deltas.

Partial state saving

- Build the component tree
- Lock it down (mark initial state)
- Subsequent modifications saved
- On restore, build component tree again
- Apply previously saved deltas
- No need to save full state!

State saving 2.0

- PartialStateHolder
  - StateHolder that can lock down state
- StateHelper
  - Manages state, tracks deltas
- No more custom saveState/restoreState
- Significantly smaller saved state!

State saving: JSF2.next

- Further optimizations?
- Better support for edge cases
- Re-execution of tags after invoke app
- Target high scalability cases
  - Fully stateless?

Controller

GET support, bookmarkable URLs,

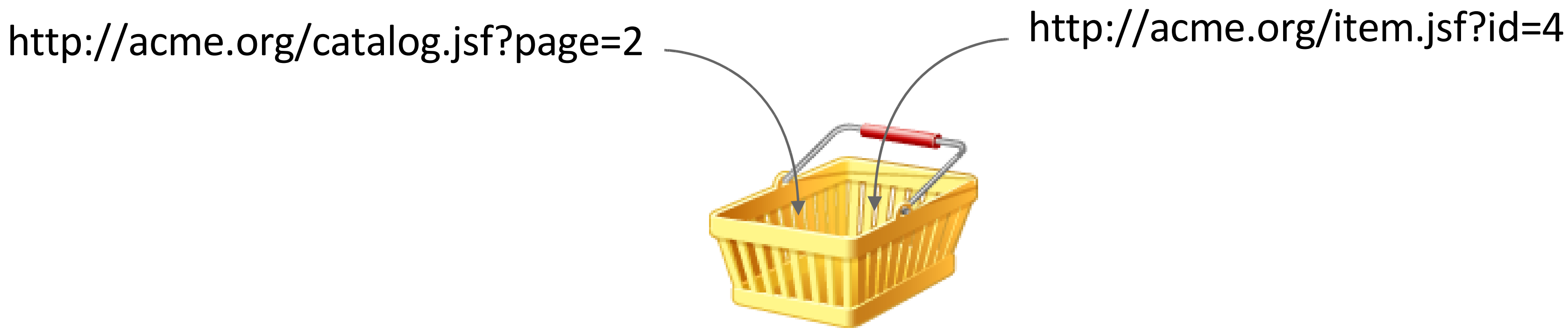
navigation and redirects,

and resource loading

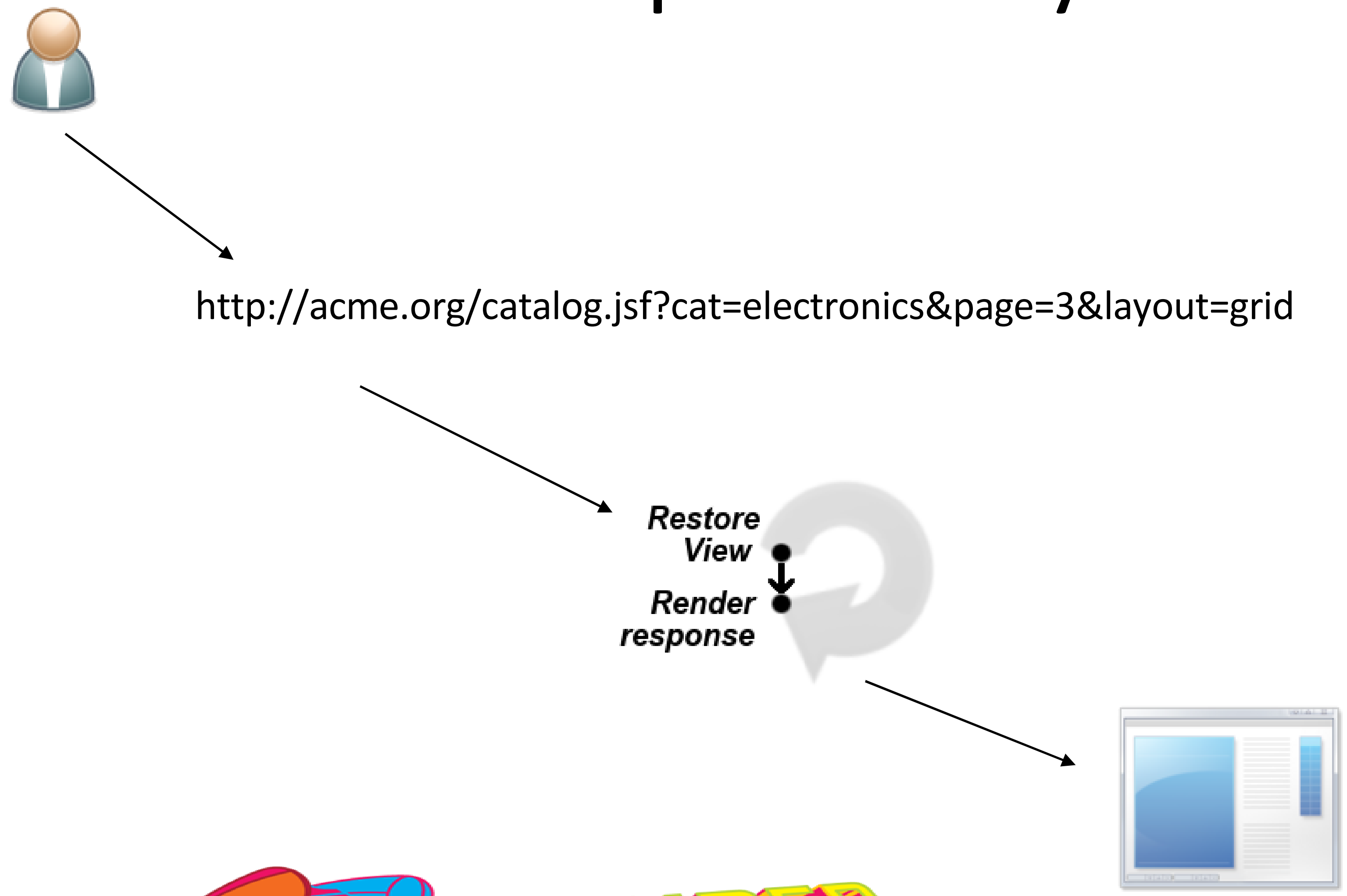
GET support

View metadata, view parameters, pre-render event listeners and bookmarkable URL components

Consuming


http://acme.org/catalog.jsf?page=2
http://acme.org/item.jsf?id=4

Initial request lifecycle


http://acme.org/catalog.jsf?cat=electronics&page=3&layout=grid
Restore View
Render response

Initial data

$$\begin{tikzcd}
\text{http://acme.org/catalog.jsf?cat=electronics\&page=3\&layout=grid} \arrow[d] \\
\text{view ID} \arrow[d] \\
\text{/catalog.xhtml}
\end{tikzcd}$$

Initial data

http://acme.org/catalog.jsf?cat=electronics&page=3&layout=grid

↓

request parameters

↓

cat=electronics
page=3
layout=grid

Bean property mapping

```
<managed-bean>
  ...
  <managed-bean-property>
    <property-name>category</property-name>
    <value>#{param['cat']}</value>
  </managed-bean-property>
</managed-bean>
```

Bean property mapping limitations

- Assignment occurs when bean is used
  - What if mapping differs based on current view?
- Implicit conversion only
  - What if property type is java.util.Date?
  - What about validation?
- What about a post-mapping listener?

eed more sophisticated, view-oriented mappin

View metadata

#{...} ⟷ XHTML

- Yet another XML schema? (YAXS!)
- Need elements for:
  - matching view ID(s)
  - describing EL binding
  - conversion
  - validation
  - post-mapping listener
  - ...

Reuse the tree

View metadata facet

```
<f:view>
  <f:metadata>
    ...
  </f:metadata>
  ...
</f:view>
```

View metadata facet

- Built-in facet of UIViewRoot
  – Known place to find metadata
  – Can be built separate from tree
- Reuses UI component infrastructure
  – Metadata is described using UI components
  – Manifests as UIPanel component
  – Easy to extend

View metadata lifecycle

- Initial request is now a full postback
  - UI component tree only contains view metadata
  - Only happens if view parameters are present
- A postback is just a postback
  - Metadata components just like any other UI components

View parameter

UIViewParameter

```
<f:view>
  <f:metadata>
    <f:viewParam name="cat" value="#{catalogBean.category}"/>
  </f:metadata>
  ...
</f:view>
```

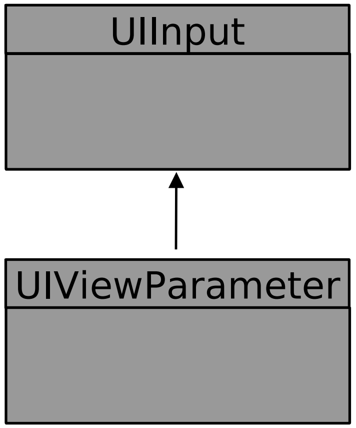
View parameter w/ converter

UIViewParameter

```
<f:view>
  <f:metadata>
    <f:viewParam name="cat" value="#{catalogBean.category}">
      <f:converter converterId="com.acme.converter.Category"/>
    </f:viewParam>
  </f:metadata>
  ...
</f:view>
```

View parameter assignment

- name – request parameter name
- value – bean property described w/ EL
- Specialization of UIInput
  - Initial value transfered from request parameter
  - Submitted value stored in component state
  - Request parameter can override value on postback
- Foundation of bookmarkable URLs



View metadata templating

```
<f:view>
  <f:metadata>
    <ui:include src="/WEB-INF/metadata/catalog.xhtml"/>
    [ or ]
    <acme:catalogMetadata/>
  </f:metadata>
  …
</f:view>
```

ore powerful & flexible than a matching patte

Post-processing

The values are set, now what?

Component system events

- Fine-grained event system in JSF 2
  - Publish/subscribe pattern (3 tiers)
- PostAddToViewEvent
  - After component is created (e.g., UIViewRoot)
- PreRenderViewEvent
  - Before component tree is rendered
  - ╫: Lifecycle :╫ if view ID is changed by listener

Post-mapping event listener

Declarative system event

```
<f:view>
  <f:metadata>
    ...
    <f:event type="preRenderView" listener="#{catalogBean.onRender}"/>
  </f:metadata>
  ...
</f:view>
```

No-args method or method that accepts `ComponentSystemEvent`

Hold the rendering!

```
public void onRender() {
  FacesContext ctx = FacesContext.getCurrentInstance();
  if (ctx.isValidationFailed() || !loadDataAttempt()) {
    ctx.getApplication().getNavigationHandler()
        .handleNavigation(ctx, null, "invalid");
  }
}
```

Force navigation if preconditions not met

Report downloads

```
<view xmlns="http://java.sun.com/jsf/core">
  <event type="preRenderView" listener="#{reportBean.download}"/>
</view>
```

Pushing the file

```
public void download() {
    FacesContext ctx = FacesContext.getCurrentInstance();
    pushFile(
        ctx.getExternalContext(),
        "/path/to/a/pdf/file.pdf",
        "file.pdf"
    );
    ctx.responseComplete();
}
```

View actions

Wouldn't it be nice if we had...?

```
<f:view>
  <f:metadata>
    ...
    <f:viewAction execute="#{catalogBean.onRender}"/>
  </f:metadata>
  ...
</f:view>
```

Including option to disable on postback

View actions

...followed by buit-in navigation?

```
<navigation-rule>
  <from-view-id>/catalog.xhtml</from-view-id>
  <navigation-case>
    <from-action>#{catalogBean.onRender}</from-action>
    <from-outcome>failure</from-outcome>
    <to-view-id>/search.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

View actions vs PreRenderView

- It's about timing
- PreRenderView
  - Executes before rendering component tree
- View action
  - Executes before building component tree
  - Why build it just to throw it away?

How do we process this URL?

http://acme.org/catalog/category/electronics

Pretty URLs

```
<rewrite-rule>
   <rewrite-view-id>/catalog.xhtml</rewrite-view-id>
   <rewrite-case>
      <url-pattern>/catalog</url-pattern>
      <url-pattern>/catalog/category/{cat}</url-pattern>
      <url-pattern>/catalog/category/{cat}/{page}</url-pattern>
   </rewrite-case>
</rewrite-rule>
```

View parameter mappings

Producing

UIOutputLink

`<h:outputLink value="/home.jsf">Home</h:outputLink>`

- Basic hyperlink-generating component
- Not aware of:
  - context path,
  - view ID extension → servlet mapping, or
  - navigation rules
- Manual query string creation
  - Does at least support <f:param>

UIOutcomeTarget

`<h:link outcome="home" value="Home"/>`

- Intelligent hyperlink-generating component
- Aware of:
  - context path,
  - uses navigation handler to derive view ID, and
  - can encode view parameters into query string
- Parameter overrides
  - Can use `<f:param>` to set parameter explicitly

Generating bookmarkable links

```
<h:link value="Previous" includeViewParams="true">
  <f:param name="page" value="#{catalogBean.previousPage}"/>
</h:link>
```

http://acme.org/catalog.jsf?q=portable+hole&page=3

/catalog.xhtml

```
<f:metadata>
  <f:viewParam name="q" value="#{catalogBean.query}"/>
  <f:viewParam name="page" value="#{catalogBean.page}"/>
</f:metadata>
```

GET support: JSF 2.next

- View actions – `<f:viewAction>`
- View restrictions – `<f:restrictView>`
- Consuming pretty URLs – `<rewrite-rules>`
- Other ideas?

Navigation

Implicit, conditional and preemptive navigation, queryable navigation rules and redirect parameters

Implicit navigation

- Fall-through case catering to prototypes
- Logical outcome => view ID
- Applies to:
  - return value of action method,
  - action of UICommand (`<h:commandButton>`),
  - outcome of UIOutcomeTarget (`<h:link>`), or
  - `NavigationHandler.handleNavigation()` method

Tweaking implicit navigation

- Can include query string
  - /product.xhtml?id=3
- Built-in directive to force a redirect
  - /product.xhtml?faces-redirect=true&id=3

A navigation shorthand

```
<h:commandButton action="#{productBean.save}" value="Save"/>
```

```
public String save() {
   // perform save logic, then...
  return "/catalog.xhtml";
}
```

A navigation short(er)hand

```
<h:commandButton action="#{productBean.save}" value="Save"/>
```

```
public String save() {
   // perform save logic, then...
  return "catalog";
}
```

Relative to current path and view ID

Can link to navigation case later

Logical outcomes aren’t logical

- Leak into business logic
- Reuse is difficult
- Void methods don’t work

Conditional navigation

- Navigation case matched based on state
- Promotes loose coupling
  - Action methods don't return "logical outcome"

Web tier ■■■■■▮ Transactional tier

- Can reduce number of navigation cases
- Navigation cases not skipped on void outcome

A conditional case

```
<navigation-case>
  <from-action>#{registration.register}</from-action>
  <if>#{currentUser.registered}</if>
  <to-view-id>/account.xhtml</to-view-id>
  <redirect include-view-params="true"/>
</navigation-case>
```

Matching a void outcome

```
<navigation-case>
  <from-action>#{catalog.search}</from-action>
  <if>#{true}</if>
  <to-view-id>/results.xhtml</to-view-id>
</navigation-case>
```

Preemptive navigation

- Evaluated at render time
- Outcome translated into bookmarkable URL
- Key elements:
  - UIOutcomeTarget (`<h:link>`, `<h:button>`)
  - implicit navigation
  - view parameters

Bookmarkable link

```
<h:link outcome="product" value="View">
  <f:param name="id" value="#{product.id}"/>
</h:link>
```

$\downarrow$

```
<a href="/product.jsf?id=3">View</a>
```

Redirect parameters

- No support in JSF 1.x
  - Made redirect after POST difficult
  - Limited usefulness of declarative navigation
- Two solutions in JSF 2
  - Explicit redirect parameters
  - View parameters

ter POST the hard way

```
FacesContext ctx = FacesContext.getCurrentInstance();
ExternalContext extCtx = ctx.getExternalContext();
String url = ctx.getApplication().getViewHandler()
   .getActionURL(ctx, "/product.xhtml") + "?id=" + getProductId();
try {
   extCtx.redirect(extCtx.encodeActionURL(url));
} catch (IOException ioe) {
   throw new FacesException(ioe);
}
```

Redirect after POST the easier way

```
<navigation-case>
  <from-action>#{productBean.save}</from-action>
  <if>#{productBean.id != null}</if>
  <to-view-id>/product.xhtml</to-view-id>
  <redirect>
    <view-param>
      <name>id</name>
      <value>#{productBean.id}</value>
    </view-param>
  </redirect>
</navigation-case>
```

Redirect after POST the best way

```
<navigation-case>
  <from-action>#{productBean.save}</from-action>
  <if>#{productBean.id != null}</if>
  <to-view-id>/product.xhtml</to-view-id>
  <redirect include-view-params="true"/>
</navigation-case>
```

Navigation: JSF 2.next

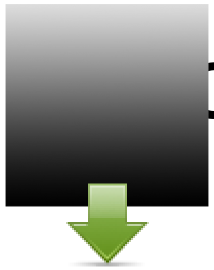
- Include view parameters automatically
- `<if>#{true}</if>` is ugly
- Navigation rules are XML hell
  - A more conscise DSL?
  - Java-based configuration?
- Other ideas?

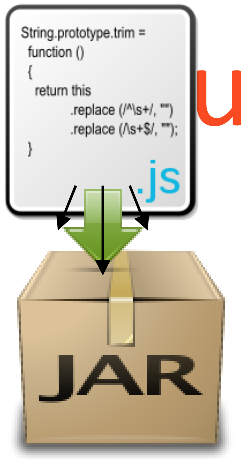
Resource handling

Native resource handling,

packaging and resource relocation

N ore “ us” servlet!





```
String.prototype.trim =
 function ()
 {
  return this
       .replace (/^\s+/, "")
       .replace (/\s+$/, "");
 }
```

Resource handling

- Load resources out of web root or JAR
- Associate resources with UIComponent
  - Resources loaded if component is rendered
- Resource loading API
- Localization

Declarative component resources

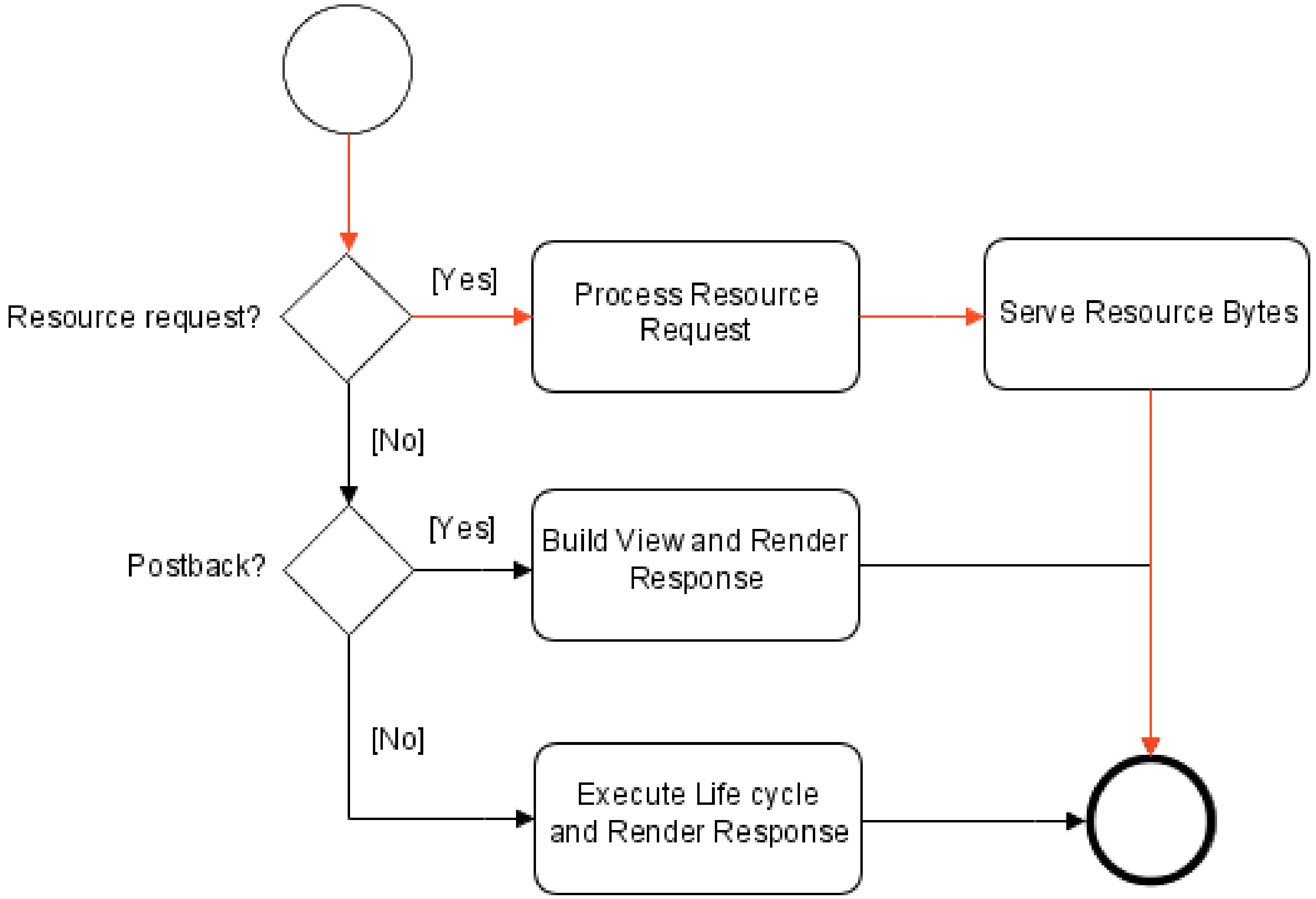
```
@ResourceDependency(
  name = "jsf.js", library = "javax.faces", target = "head")
public class MyComponent extends UIOutput { ... }
```

A resource at a glance

- Structure
  - Name
  - Library
  - Locale
  - Version

- Packaging
  - Web root
    - `/resources`
  - Classpath
    - `META-INF/resources`

A third request processing scenario



Resolving a resource

localePrefix/libraryName/libraryVersion/resourceName/resourceVersion

Path segments in grey are optional

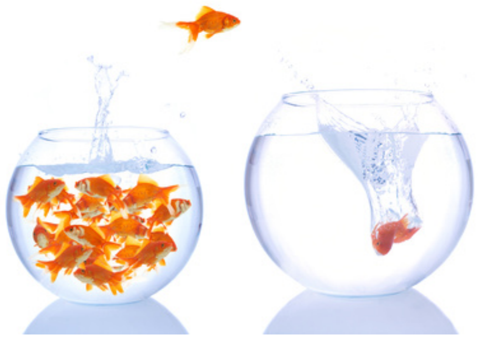
- Served from web root

```
<h:graphicImage name="visa.png"/>
```

- Served from classpath of creditcards.jar

```
<h:graphicImage name="visa.png" library="creditcards"/>

<h:graphicImage value="#{resources['creditcards:visa.png']}"/>
```

Resource relocation



- Resources can target section of document
- Essential for templating

```
<html>
  <h:head>
    <title>Resource Relocation Example</title>
  </h:head>
  <h:body>
    <h:outputScript name="script.js" target="head"/>
  </h:body>
</html>
```

Resources: JSF 2.next

- Sprite generation
- Compression support
- What else?

Model

Java EE 6 component model,

Bean Validation, error handling

and resource loading

Java EE 6: Goals

- Extensibility
  - Allow more components to be standalone (EJB 3.1)
- Profiles
  - Subsets of “full” EE platform
  - Web Profile
- Pruning
  - CMP, JAX-RPC, JAXR, JSR-88 are “pruned” in EE6
- Technology Improvements

Java EE 6: Newcomers

- Managed Beans (part of JSR-316)
- Contexts and Dependency Injection - JSR-299
- Bean Validation - JSR-303
- JAX-RS (RESTful Web Services) - JSR-311

Java EE 6: Notable updates

- Servlet 3.0
- JPA 2.0
- Type-safe Criteria API
- Extra mappingsEJB 3.1
- No-interface views
- – Package in wars
- – Async and timer support
- – Embeddable
- – Embeddable

...and JSF 2.0, of course!

Web profile contents

Persistence

- JPA 2.0
- JTA

Component model

- EJB 3.1 Lite
- Bean Validation
- CDI (JSR-299)

Presentation

- JSF 2.0
- Servlet 3.0

JSR-299: Essential ingredients

- Beans types
- Qualifier annotations
- Scope
- Alternatives
- An EL name (optional)
- Interceptors and decorators
- The implementation

Simple example

```
public class Hello {
   public String sayHello(String name) {
      return "Hello, " + name;
   }
}
```

Any Managed Bean can use CDI services

```
@Stateless
public class Hello {
   public String sayHello(String name) {
      return "Hello, " + name;
   }
}
```

...even EJBs!

Simple example

```
public class Printer {

  @Inject Hello hello;

  public void printHello() {
    System.out.println(hello.sayHello("Devoxx"));
  }
}
```

`@Inject` defines injection point, assumes `@Default` qualifier

Constructor injection

```
public class Printer {
  private Hello hello;

  @Inject
  public Printer(Hello hello) { this.hello = hello; }

  public void printHello() {
    System.out.println(hello.sayHello("Devoxx"));
  }
}
```

`@Inject` marks constructor to be called by container; arguments injected automatically

Bean EL names

```
@Named("hello")
public class Hello {
  private String name; // getters and setters not shown
  public void sayHello() {
    System.out.println("Hello, " + name);
  }
}
```

`@Named` makes bean available to EL

```
@Named
public class Hello {
  …
}
```

Name can be defaulted to simple name of class

JSF view

Invoking a bean via EL

```
<h:inputText value="#{hello.name}"/>
<h:commandButton value="Say Hello" action="#{hello.sayHello}"/>
```

Qualifier

An annotation that lets a client choose between multiple implementations of an API at runtime

Write a qualified implementation

```
@Casual
public class Hi extends Hello {
  public String sayHello(String name) {
    return "Hi, " + name;
  }
}
```

This second Hello bean is qualified `@Casual`

Using a qualifier

```
public class Printer {

  @Inject @Casual Hello hello;

  public void printHello() {
    System.out.println(hello.sayHello("Devoxx"));
  }
}
```

Injects the `@Casual` implementation of Hello

Scopes and contexts

- Built-in scopes:
- Any servlet request: `@ApplicationScoped`, `@RequestScoped`, `@SessionScoped`
- JSF requests - `@ConversationScoped`
- Dependent scope (Default): `@Dependent`
- Custom scopes
  - Define scope type annotation (e.g., `@FlashScoped`)
  - Context impl defines where bean is stored

Producer methods

- Producer methods allow control over bean creation where:
  - the objects to be injected are not managed instances
  - the concrete type of the objects to be injected may vary at runtime
  - the objects require some custom initialization that is not performed by the bean constructor

Parameterized EL methods

- Syntax similar to Java method calls
- Method arguments are EL expressions
- Arguments resolved at different times:
  - Value expression: at render time
  - Method expression: when event is fired

```
<h:commandButton action="#{hello.sayHello('Devoxx')}" .../>
    <h:commandButton action="#{hello.sayHello(currentConference)}" .../>
```

Validation

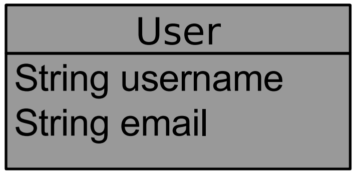
Bean Validation integration,

validating empty fields and multi-field

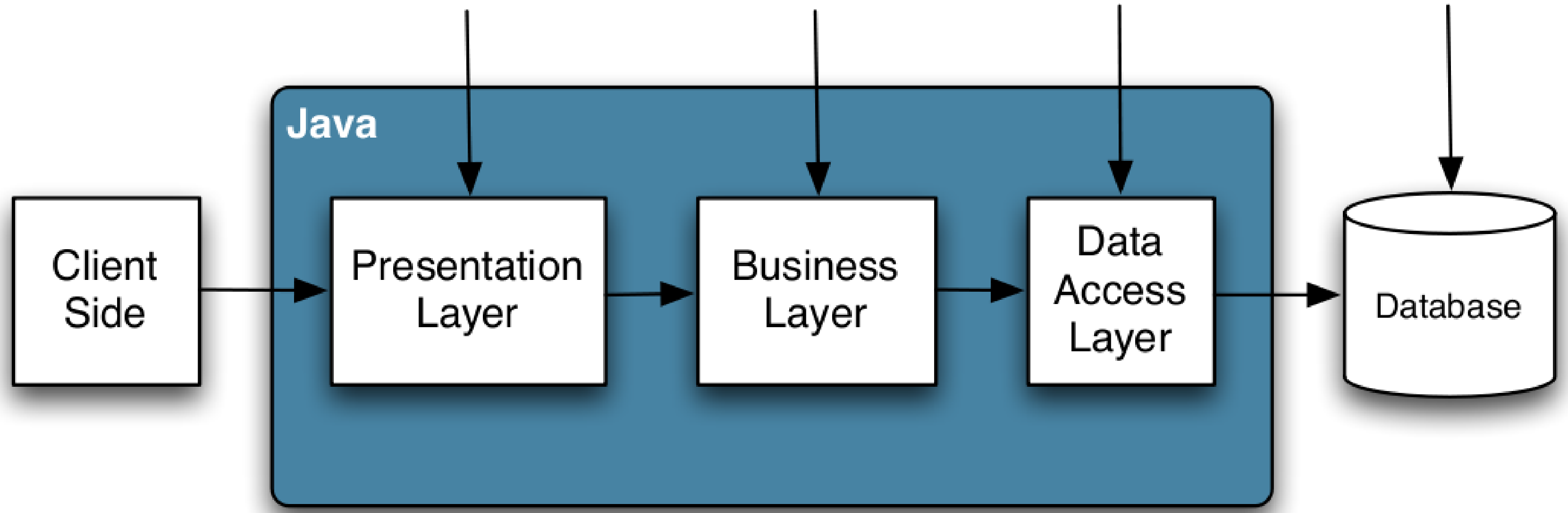
validation with post-validate events

Constraints in the enterprise

One model...



...validated across multiple layers



Bean Validation (JSR-303)

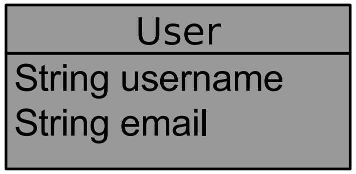
- Constrain once, validate anywhere
- Centrally define constraints in model class
  - Constraints described using annotations
- JSF integration
  - Enforce constraints in presentation layer
  - Replaces existing JSF validators
  - Zero configuration!

Defining constraints on the model

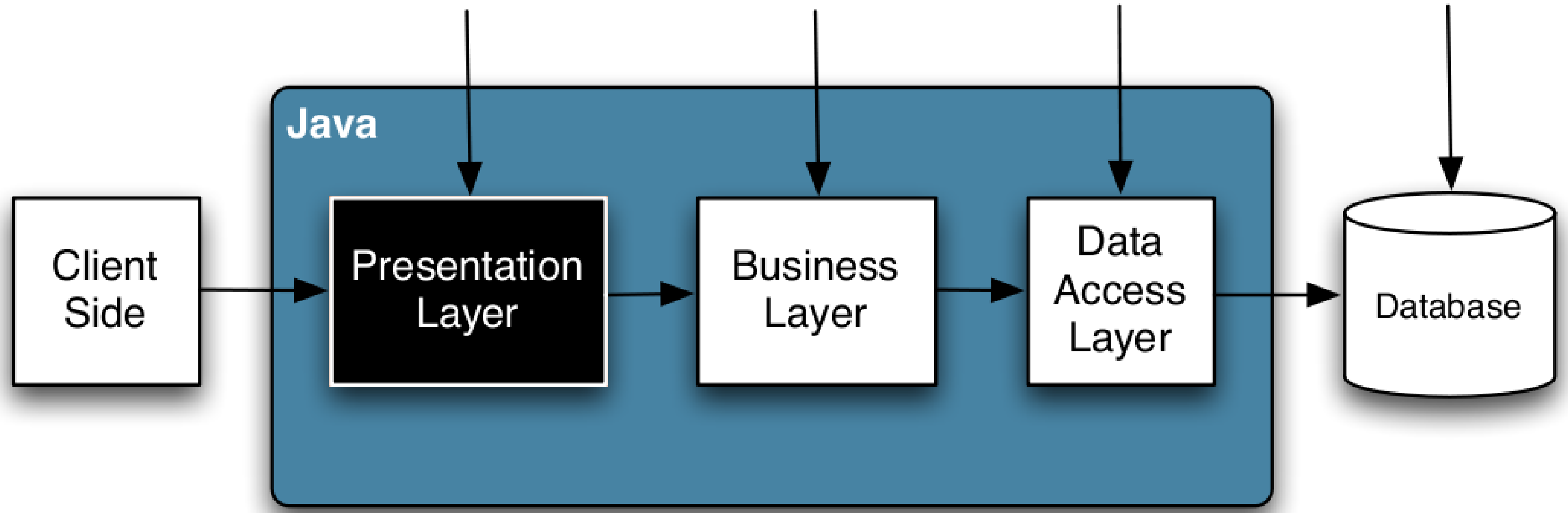
```
public class User {
   ...
   @NotNull @Size(min = 3, max = 25)
   public String getUsername() { return username; }

   @NotNull @Email
   public String getEmail() { return email; }
}
```

Constraints in JSF

One model...



...validated across multiple layers



Enforcing constraints in the UI

```
<h:inputText id="username" value="#{user.username}"/>
```

```
<h:inputText id="email" value="#{user.email}"/>
```

Zeroconf!

Constraining partially

```
<h:inputText id="username" value="#{user.username}">
  <f:validateBean disabled="true"/>
</h:inputText>
```

```
<f:validateBean validationGroups="com.acme.BareMinimum">
  <h:inputText id="email" value="#{user.email}"/>
</:validateBean>
```

The case of the empty field

- Validation skipped if value is:
  - null
  - a zero-length string
- Unless...
  - Bean Validation is present or
  -

```
<context-param>
  <param-name>javax.faces.VALIDATE_EMPTY_FIELDS</param-name>
  <param-value>true</param-value>
</context-param>
```

Do you mean null?

- Problem: user can't enter null in text field
- Side-effect: inadvertent database updates
- Solution: interpret empty strings as null

```
<context-param>
  <param-name>
    javax.faces.INTERPRET_EMPTY_STRING_SUBMITTED_VALUES_AS_NULL
  </param-name>
  <param-value>true</param-value>
</context-param>
```

Multi-field validation

- A tougher problem than it seems
- Two approaches:

Before model update	After model update
– Compare UIInput values	– Validate populated model
– PostValidateEvent	– Bean Validation

Listening for post validate

```
<h:form>
  <f:event type="postValidate" listener="#{minMax.validate}"/>
  <h:inputText id="min" value="#{bean.min}"
    binding="#{minMax.minInput}"/>
  <h:inputText id="max" value="#{bean.max}"
    binding="#{minMax.maxInput}"/>
  <h:commandButton value="Submit"/>
</h:form>
```

Validating across fields

```
@Inject FacesContext ctx;
private UIInput minInput, maxInput; // accessors hidden
public void validate() {
  if (ctx.isValidationFailed()) { return; }
  if ((Integer) maxInput.getValue() < (Integer) minInput.getValue()) {
    ctx.addMessage(maxInput.getClientId(ctx),
      new FacesMessage("cannot be less than min value"));
    ctx.validationFailed();
    ctx.renderResponse();
  }
}
```

Validation JSF.next

- What about postModelUpdate?
- Adding FacesMessages is tedious
- Graph Validation (Bean Validation on object)

Error handling

Exception handlers, exception events, servlet errors and the default error page

The good news

No more swallowed exceptions!

The bad news

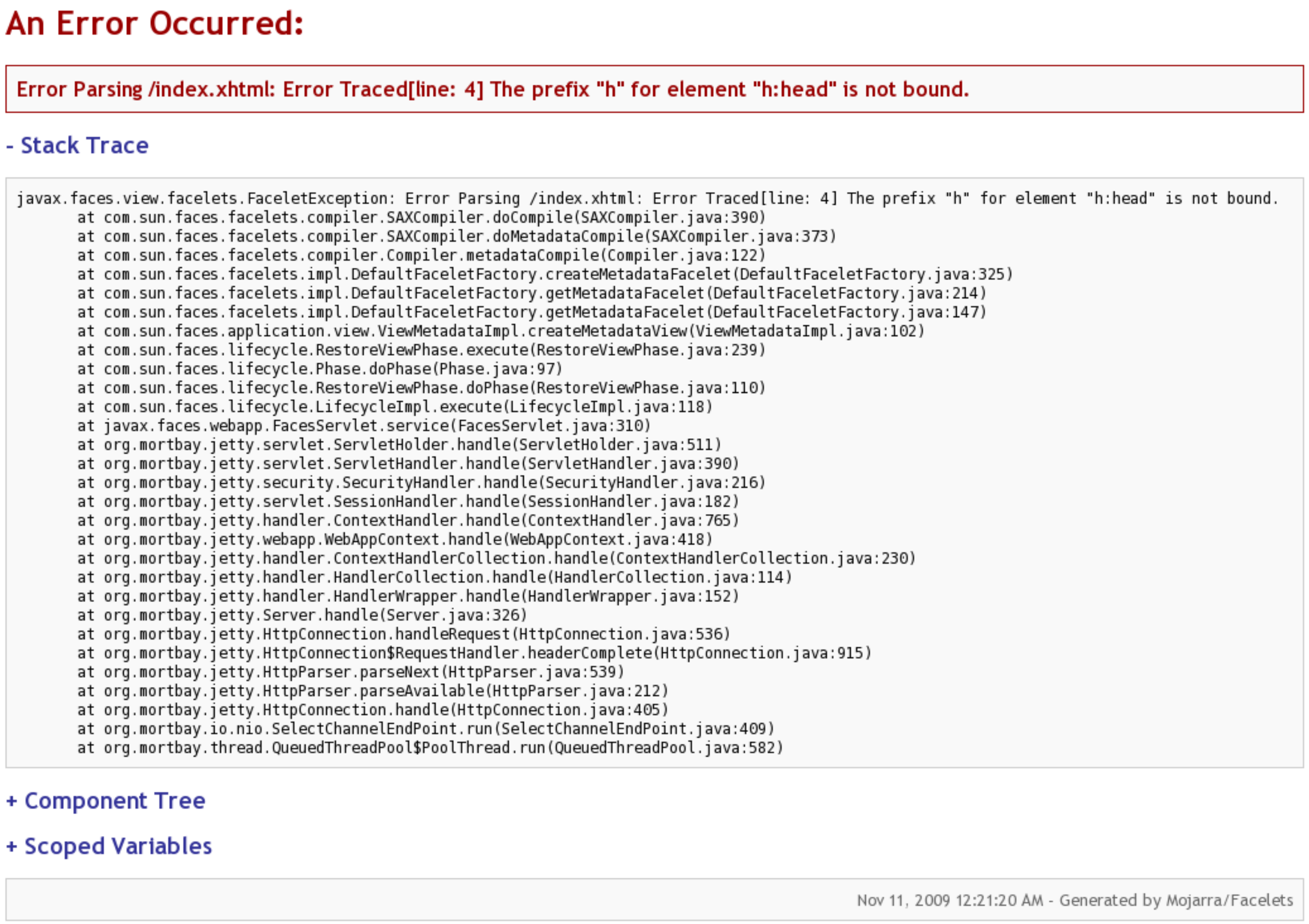
You're still going to get exceptions

Exception handler

Ugh!

- Hub for handling unexpected exceptions
- When exception is thrown:
  - ExceptionQueuedEvent is published
  - Exception handler queues exception
- After each phase:
  - Exception handler unwraps first exception, rethrows as FacesException

Default error page



An Error Occurred:

Error Parsing /index.xhtml: Error Traced[line: 4] The prefix "h" for element "h:head" is not bound.

- Stack Trace

```
javax.faces.view.facelets.FaceletException: Error Parsing /index.xhtml: Error Traced[line: 4] The prefix "h" for element "h:head" is not bound.
	at com.sun.faces.facelets.compiler.SAXCompiler.doCompile(SAXCompiler.java:390)
	at com.sun.faces.facelets.compiler.SAXCompiler.doMetadataCompile(SAXCompiler.java:373)
	at com.sun.faces.facelets.compiler.Compiler.metadataCompile(Compiler.java:122)
	at com.sun.faces.facelets.impl.DefaultFaceletFactory.createMetadataFacelet(DefaultFaceletFactory.java:325)
	at com.sun.faces.facelets.impl.DefaultFaceletFactory.getMetadataFacelet(DefaultFaceletFactory.java:214)
	at com.sun.faces.facelets.impl.DefaultFaceletFactory.getMetadataFacelet(DefaultFaceletFactory.java:147)
	at com.sun.faces.application.view.ViewMetadataImpl.createMetadataView(ViewMetadataImpl.java:102)
	at com.sun.faces.lifecycle.RestoreViewPhase.execute(RestoreViewPhase.java:239)
	at com.sun.faces.lifecycle.Phase.doPhase(Phase.java:97)
	at com.sun.faces.lifecycle.RestoreViewPhase.doPhase(RestoreViewPhase.java:110)
	at com.sun.faces.lifecycle.LifecycleImpl.execute(LifecycleImpl.java:118)
	at javax.faces.webapp.FacesServlet.service(FacesServlet.java:310)
	at org.mortbay.jetty.servlet.ServletHolder.handle(ServletHolder.java:511)
	at org.mortbay.jetty.servlet.ServletHandler.handle(ServletHandler.java:390)
	at org.mortbay.jetty.security.SecurityHandler.handle(SecurityHandler.java:216)
	at org.mortbay.jetty.servlet.SessionHandler.handle(SessionHandler.java:182)
	at org.mortbay.jetty.handler.ContextHandler.handle(ContextHandler.java:765)
	at org.mortbay.jetty.webapp.WebAppContext.handle(WebAppContext.java:418)
	at org.mortbay.jetty.handler.ContextHandlerCollection.handle(ContextHandlerCollection.java:230)
	at org.mortbay.jetty.handler.HandlerCollection.handle(HandlerCollection.java:114)
	at org.mortbay.jetty.handler.HandlerWrapper.handle(HandlerWrapper.java:152)
	at org.mortbay.jetty.Server.handle(Server.java:326)
	at org.mortbay.jetty.HttpConnection.handleRequest(HttpConnection.java:536)
	at org.mortbay.jetty.HttpConnection$RequestHandler.headerComplete(HttpConnection.java:915)
	at org.mortbay.jetty.HttpParser.parseNext(HttpParser.java:539)
	at org.mortbay.jetty.HttpParser.parseAvailable(HttpParser.java:212)
	at org.mortbay.jetty.HttpConnection.handle(HttpConnection.java:405)
	at org.mortbay.io.nio.SelectChannelEndPoint.run(SelectChannelEndPoint.java:409)
	at org.mortbay.thread.QueuedThreadPool$PoolThread.run(QueuedThreadPool.java:582)
```

+ Component Tree

+ Scoped Variables

Nov 11, 2009 12:21:20 AM - Generated by Mojarra/Facelets

Development diagnostics

/javax.faces.error.xhtml

- Activated when ProjectStage = Development
- Report includes:
  - stack trace of exception
  - UI component tree
  - scoped variables
  - view ID and line number
  - anything else?

Bubbling over in production

- Exceptions ↷ servlet error handler (web.xml)

```
<error-page>
  <exception-type>com.acme.SecurityException</exception-type>
  <location>/accessDenied.jsf</location>
</error-page>
```

- Several problems:
  - Error page is outside of JSF life cycle
  - Error page must include servlet mapping
  - Context of request is left behind

Declarative error handling in JSF

Wouldn’t it be nice if we had...?

```
<exception class="javax.persistence.EntityNotFoundException">
  <redirect view-id="/error/404.xhtml">
    <message severity="warn">Record not found</message>
  </redirect>
</exception>
```

Ajax error handling

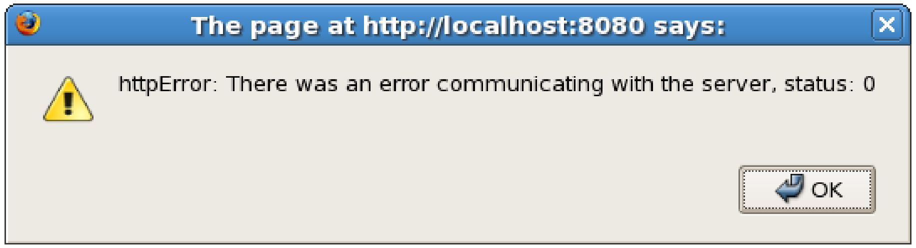
- JavaScript error callback for single request

```
<f:ajax ... onerror="handle_specific_error"/>
```

- Global JavaScript error listener

```
jsf.ajax.addOnError(handle_all_errors);
```

- Alert window fallback in development


The page at http://localhost:8080 says:

httpError: There was an error communicating with the server, status: 0

Pain relief

Select items from collections,

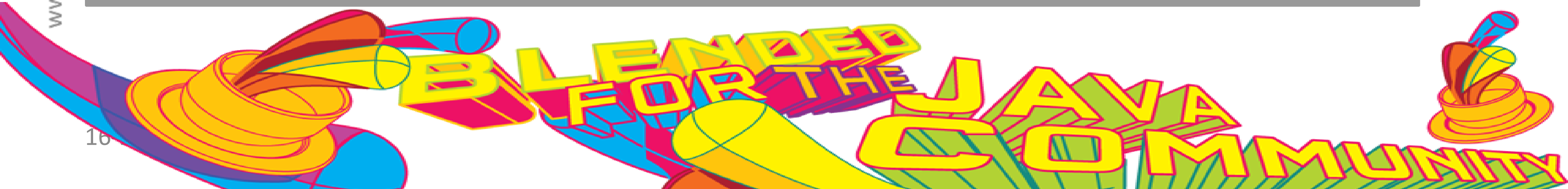
validation failed flag, API improvements,

varStatus on ui:repeat, and more...

From collection to select items

```
<h:selectOneMenu value="#{product.category}">
  <f:selectItems value="#{catalogBean.categories}" var="cat"
    itemLabel="#{cat.name}" itemValue="#{cat}"
    noSelectionValue="#{catalogBean.defaultCatalog}"/>
</h:selectOneMenu>
```

```
@Named
public class CatalogBean {
   public List<Category> getCategories() {
      return ...;
  }
}
```



Minor improvements that add up

- Retrieve faces messages as java.util.List
  - FacesContext.getMessageList()
  - FacesContext.getMessageList(String clientId)
- Preserve faces messages across redirect
  - ExternalContext.getFlash().setKeepMessages(true)
- Flag indicating whether validation failed
  - FacesContext.isValidationFailed()
- ActionEvent optional for action listeners

Pain relief: JSF 2.next

- UIData components
  - java.util.Collection
  - varStatus
  - row state
- Standard components
  - h:fileUpload
  - Separate spec?
- Facelets from JAR
- EL
  - Static methods
  - Enum support
- Rendered attribute
- Generated ids
- Container injection

Community

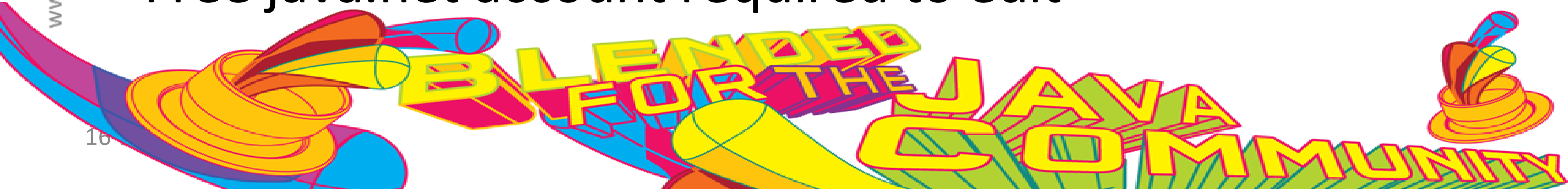
JSR-314-OPEN mailinglist,

javaserverfaces-spec-public project,

JCP.org and you!

Steps towards openness

- Semi-public mailinglist – JSR-314-OPEN
  - http://archives.java.sun.com/jsr-314-open.html
  - Free registration required to view
  - Must be EG member to post
- Public issue tracker – java.net project
  - https://javaserverfaces-spec-public.dev.java.net
  - No registration required to view
  - Free java.net account required to edit



Next steps

- Anonymous read access to JSR-314-OPEN
  - Allow community to follow along
  - Make sharing links easier
  - Indexable by search engines

Google bing
Nabble

- Non-EG member invites to JSR-314-OPEN
  - Prime candidates – implementation team members
- Read-write community mailinglist

Creating a JCP.org profile

Did you know that anyone can have a JCP.org profile?

Just sign up!

JCP.org 2.0 - Launched June 2009

- Goals are to enhance:
  - participation,
  - communication, and
  - transparency
- Personalized content
- Discussion boards
- Wiki

Becoming a JCP member

Did you know that anyone can become a JCP member?

Just sign the JSPA!

JCP membership fee (JSPA)

- Commercial organizations: $5000
- Educational/non-profit organizations: $2000
- Java User Groups (JUGs): free!
- Individuals: free!

Membership benefits

- Submit JSRs
- Serve on a JSR Expert Group (EG)
- Vote in EC elections (reps who vote on specs)
- http://jcp.org/en/participation/committee
- View EC meeting minutes

JSF community home page

http://javaserverfaces.org (future)

Single entry point into the JSF ecosystem:

- Specification and API docs
- Mailinglists and forums
- Issue tracker
- FAQs and guides
- Implementations, component libraries

Summary

- JSF 2 is a drastic improvement
- Embraced de-facto community standards
- JSR-314 seeks to be role model for openness
- Still lots of room for innovation in JSF 2.next
- You can be part of the process!

See you at the JSF 2 BOF! (20:00)

Learn

- http://tinyurl.com/jsf2new
- http://tinyurl.com/jsf2devworks
- http://tinyurl.com/jsf2dzone
- http://tinyurl.com/jsf2driscoll
- http://tinyurl.com/jsf2ryan

Try

- http://tinyurl.com/jsf2ri
- http://tinyurl.com/jsf2issue