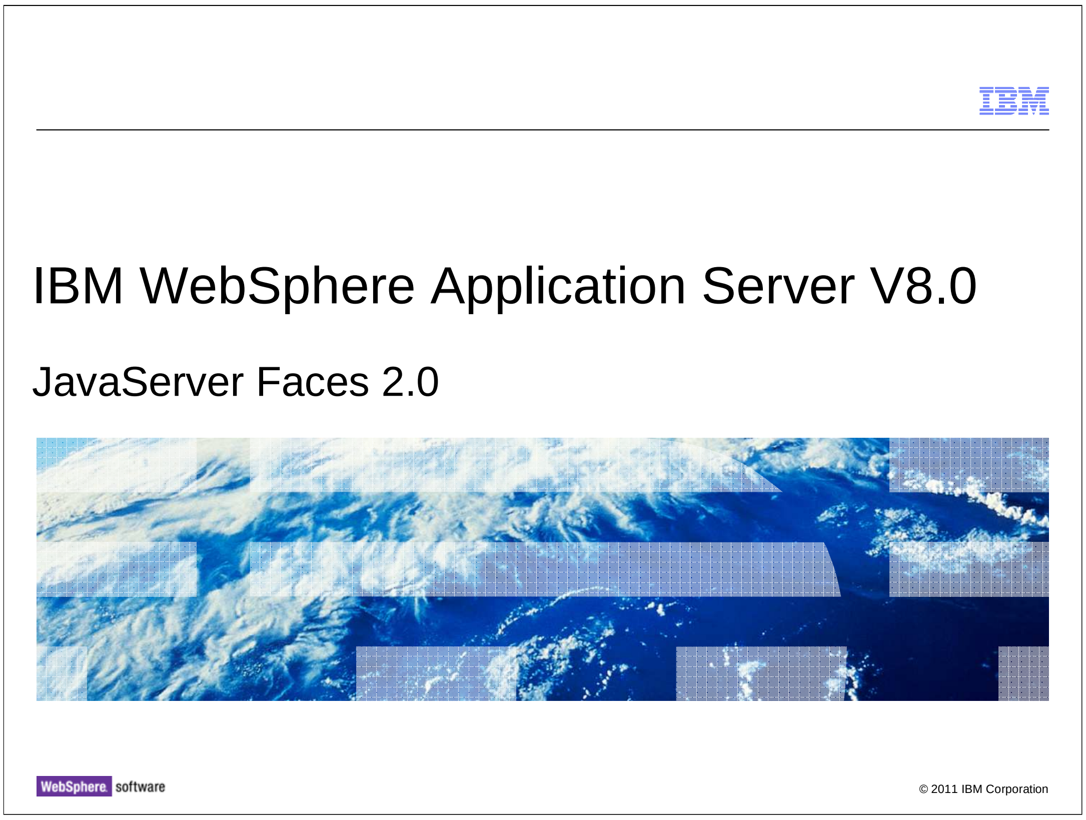


IBM WebSphere Application Server V8.0

JavaServer Faces 2.0

This presentation describes support for JavaServer Faces (JSF) 2.0 for Java EE included in IBM WebSphere Application Server V8.

Table of contents

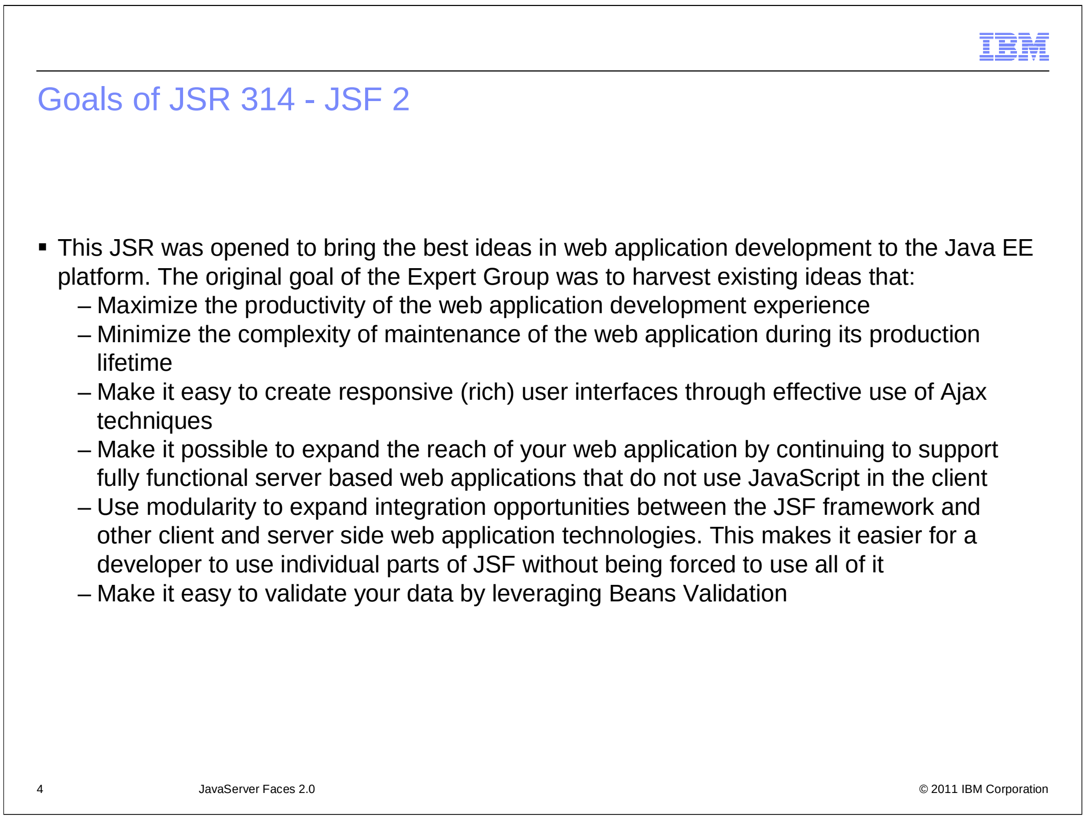
This session will provide an overview of JSF 2.0, scenarios of JSF 2.0, and end with a summary and references of JSF 2.0.

Section

Overview

3 JavaScript Faces 2.0 © 2011 IBM Corporation

This section will discuss JSF 2.0.



Goals of JSR 314 - JSF 2

- This JSR was opened to bring the best ideas in web application development to the Java EE platform. The original goal of the Expert Group was to harvest existing ideas that:
  - Maximize the productivity of the web application development experience
  - Minimize the complexity of maintenance of the web application during its production lifetime
  - Make it easy to create responsive (rich) user interfaces through effective use of Ajax techniques
  - Make it possible to expand the reach of your web application by continuing to support fully functional server based web applications that do not use JavaScript in the client
  - Use modularity to expand integration opportunities between the JSF framework and other client and server side web application technologies. This makes it easier for a developer to use individual parts of JSF without being forced to use all of it
  - Make it easy to validate your data by leveraging Beans Validation

4 JavaServer Faces 2.0 © 2011 IBM Corporation

JSR-314 JavaServer Faces 2.0 was opened to bring the best ideas in web application development to the Java EE platform. The original goal of the JSF 2.0 Expert Group was to harvest existing ideas that fell into the categories seen in this slide.

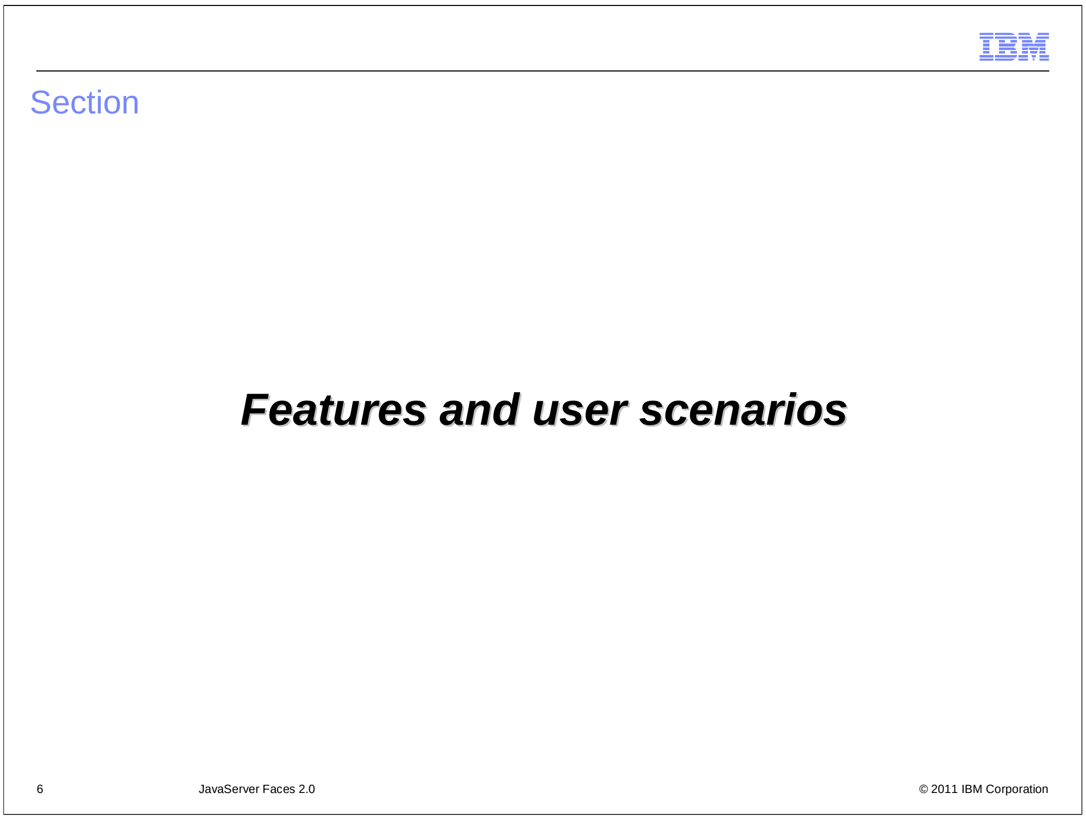
The end result is a specification that offers many features that will help increase developer productivity, decrease memory consumption, and standardized some of the best features from well known component libraries.

JSF is continuing to improve the standard to carry it into the future as a strong choice for developing UIs.

What is JSF 2

5 JavaServer Faces 2.0 © 2011 IBM Corporation

There are a lot of new features added to JSF 2.0. They can be broken down into four major themes: Ease of Development, Performance, Technology Adoption and General Fixes. This presentation will take a closer look at each one of these features listed here and show different user scenarios for each one.



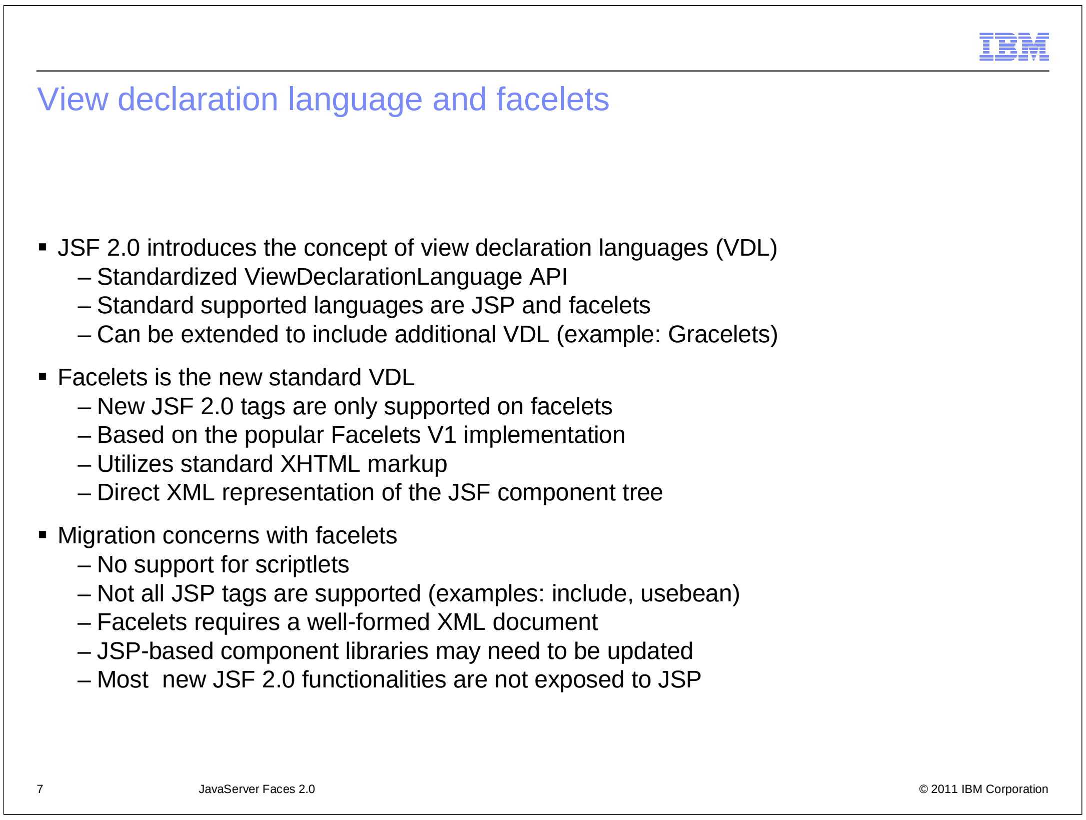
Section

Features and user scenarios

6 JavaServer Faces 2.0 © 2011 IBM Corporation

Next this session will discuss some usage scenarios.

Each scenario will touch on a feature from the four themes defined in the overview.



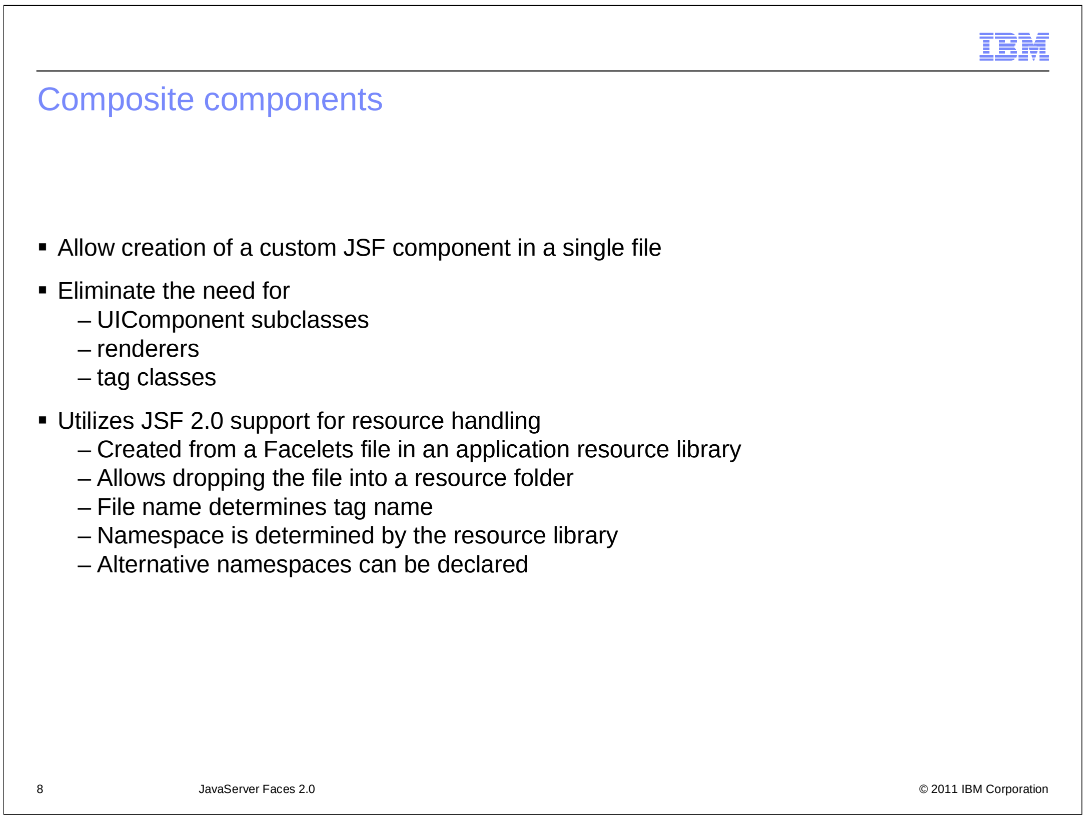
View declaration language and facelets

- JSF 2.0 introduces the concept of view declaration languages (VDL)
  - Standardized ViewDeclarationLanguage API
  - Standard supported languages are JSP and facelets
  - Can be extended to include additional VDL (example: Gracelets)
- Facelets is the new standard VDL
  - New JSF 2.0 tags are only supported on facelets
  - Based on the popular Facelets V1 implementation
  - Utilizes standard XHTML markup
  - Direct XML representation of the JSF component tree
- Migration concerns with facelets
  - No support for scriptlets
  - Not all JSP tags are supported (examples: include, usebean)
  - Facelets requires a well-formed XML document
  - JSP-based component libraries may need to be updated
  - Most new JSF 2.0 functionalities are not exposed to JSP

7 JavaServer Faces 2.0 © 2011 IBM Corporation

In order to move away from using JSP pages, JSF introduced the notion of a View Declaration Language (VDL) starting with JSF 2.0. Using a standardized API, developers can create VDLs for arbitrary languages while maintaining support for existing JSF 2.0 tag libraries. By default, JSF 2.0 supports JSP and Facelets as its VDLs.

Of these two VDLs, Facelets is considered the standard going forward (several tags introduced in JSF 2.0 are only compatible with Facelets). The Facelets implementation used in JSF 2.0 is based on the previously third-party Facelets library. With Facelets, pages are authored in XML, which imposes some additional constraints on developers. Namely, all Facelets pages must be well-formed XMLs and support several JSP features and tags that are lacking (such as scriptlets and various JSP tags).

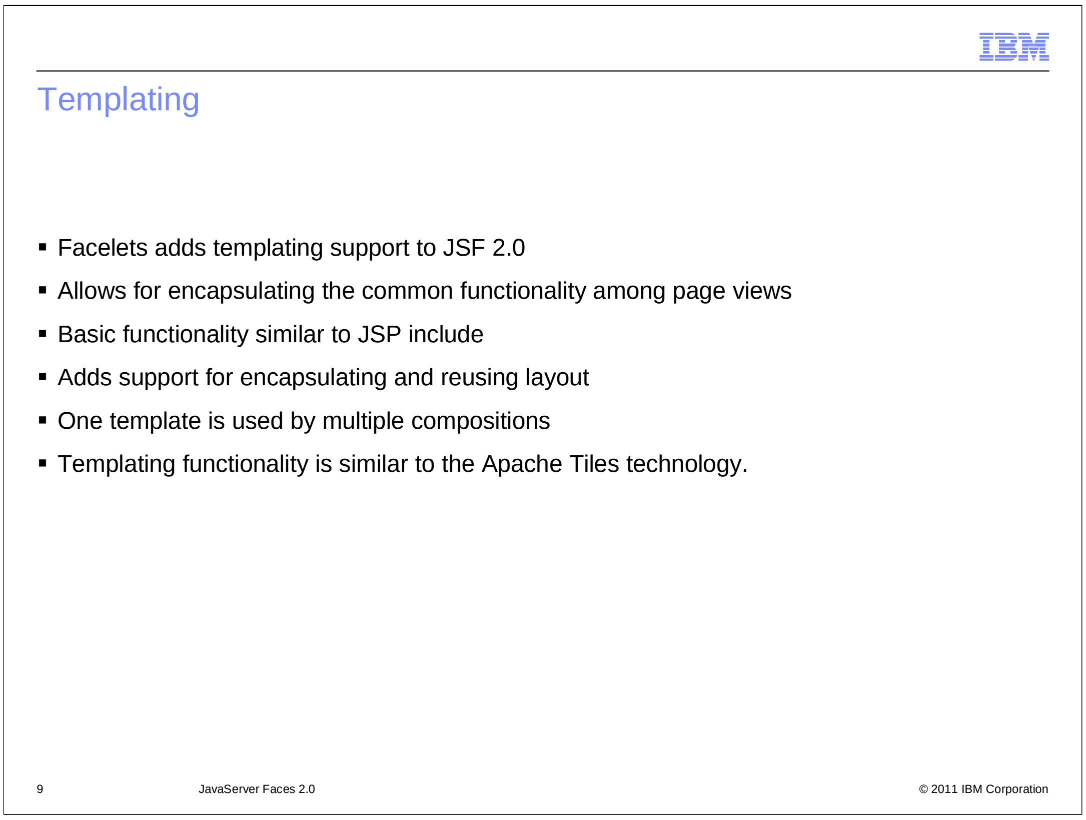


Composite components

- Allow creation of a custom JSF component in a single file
- Eliminate the need for
  - UIComponent subclasses
  - renderers
  - tag classes
- Utilizes JSF 2.0 support for resource handling
  - Created from a Facelets file in an application resource library
  - Allows dropping the file into a resource folder
  - File name determines tag name
  - Namespace is determined by the resource library
  - Alternative namespaces can be declared

8 JavaServer Faces 2.0 © 2011 IBM Corporation

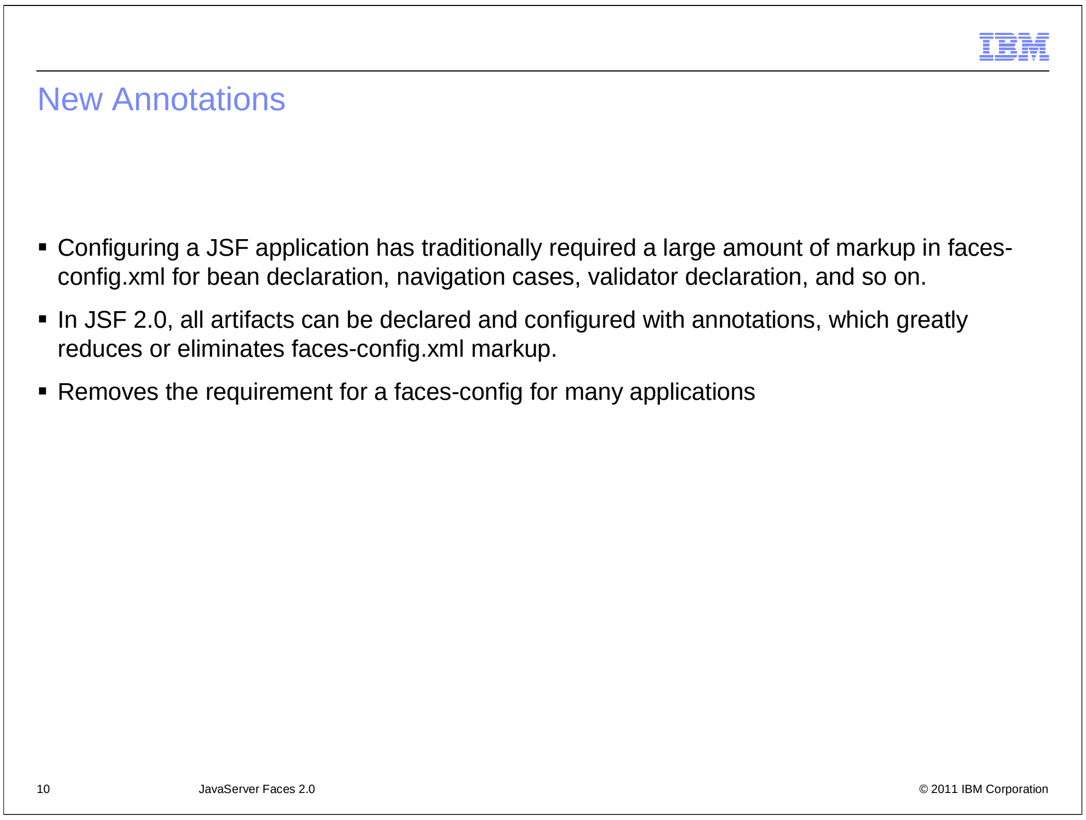
Composite components are a new JSF 2.0 feature that allow for greatly simplified development of JSF components. Creating a composite component is just a matter of creating an XHTML file and placing it in any required resource directory. The convention for composite components is that the file name determines the tag name and the resource directory determines the namespace.



Templating

- Facelets adds templating support to JSF 2.0
- Allows for encapsulating the common functionality among page views
- Basic functionality similar to JSP include
- Adds support for encapsulating and reusing layout
- One template is used by multiple compositions
- Templating functionality is similar to the Apache Tiles technology.

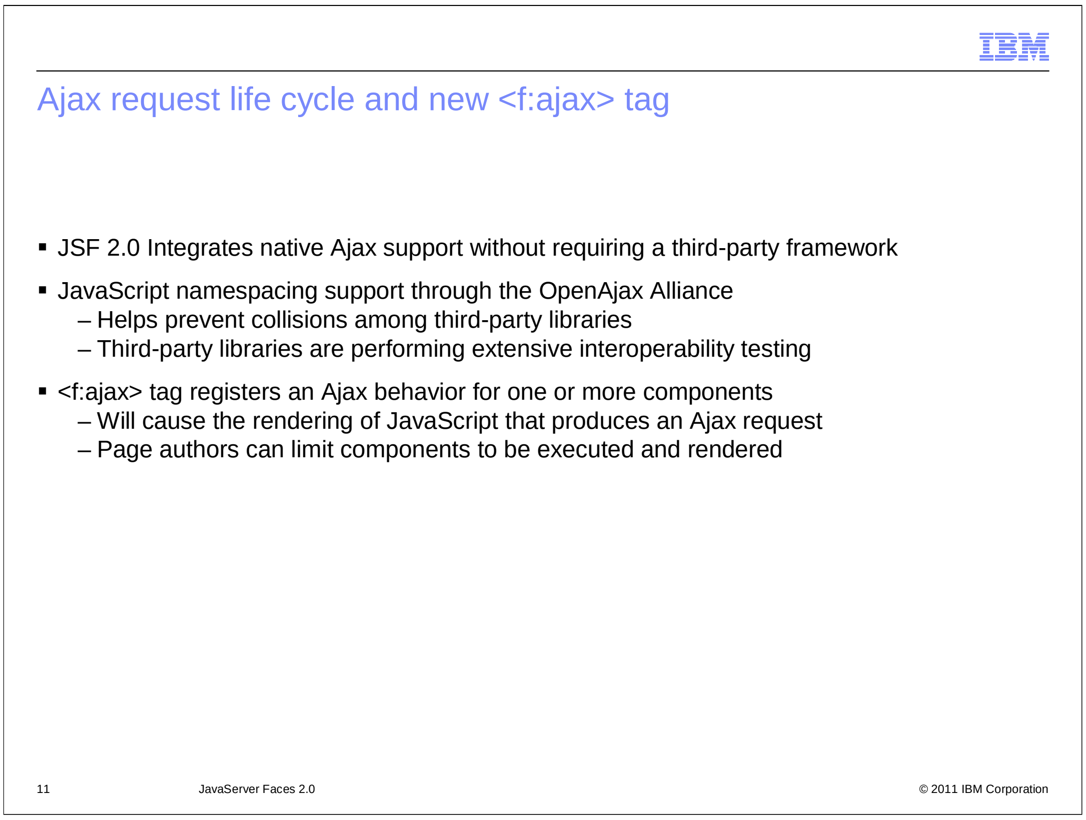
Templating allows JSF 2.0 applications to separate commonly-repeated portions of pages into separate pages that can be included within other pages, similar to a JSP include. However, unlike a simple JSP include, JSF 2.0 templates can be parameterized, much like with Apache Tiles.



New Annotations

- Configuring a JSF application has traditionally required a large amount of markup in faces-config.xml for bean declaration, navigation cases, validator declaration, and so on.
- In JSF 2.0, all artifacts can be declared and configured with annotations, which greatly reduces or eliminates faces-config.xml markup.
- Removes the requirement for a faces-config for many applications

Typical JSF applications require a large amount of configuration in the form of a faces-config.xml descriptor. JSF 2.0 introduces a large number of annotations that allow such configuration to be written inline with components, validators, beans, and so on. For most applications, the new JSF 2.0 annotations eliminate the need to maintain a separate faces-config.xml descriptor.



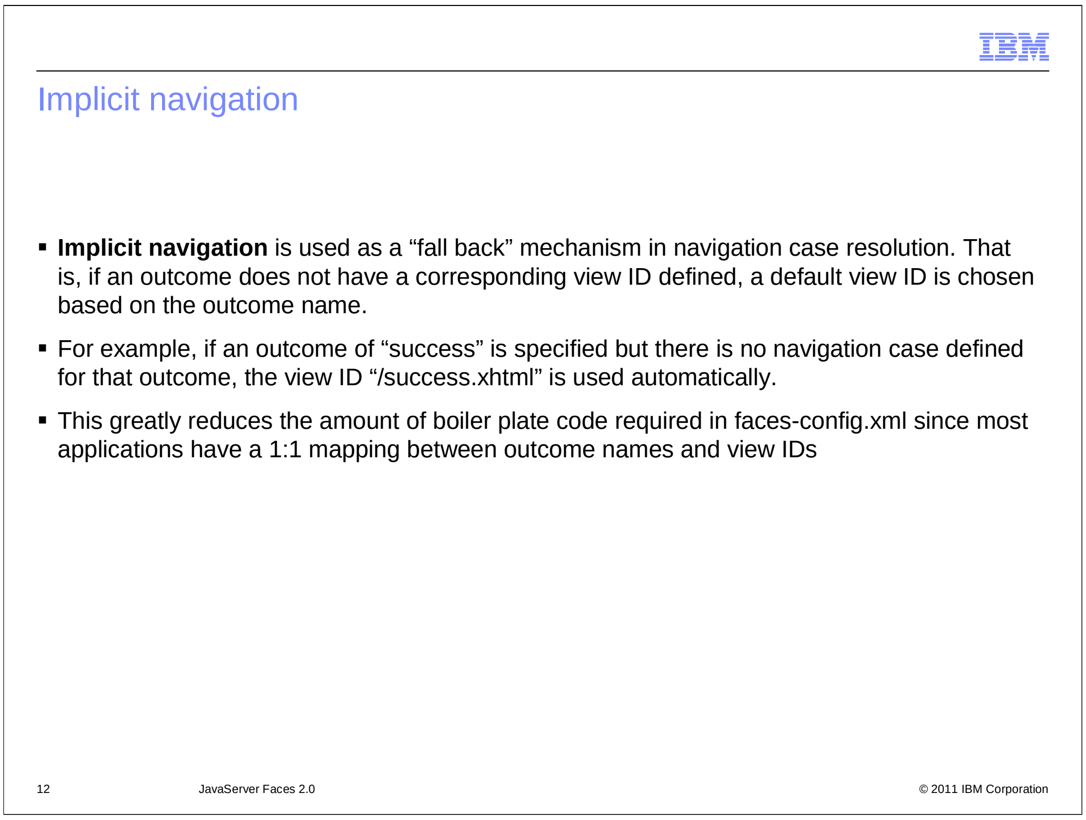
Ajax request life cycle and new <f:ajax> tag

- JSF 2.0 Integrates native Ajax support without requiring a third-party framework
- JavaScript namespacing support through the OpenAjax Alliance
  - Helps prevent collisions among third-party libraries
  - Third-party libraries are performing extensive interoperability testing
- <f:ajax> tag registers an Ajax behavior for one or more components
  - Will cause the rendering of JavaScript that produces an Ajax request
  - Page authors can limit components to be executed and rendered

11 JavaServer Faces 2.0 © 2011 IBM Corporation

New to JSF 2.0 is native support for Ajax as a standard behavior for all components. By including an <f:ajax> tag in or around a component or set of components, Ajax is enabled for the specified event types. Optionally, developers can limit which components caused backend execution or frontend rendering to occur for Ajax-enabled events.

JSF 2.0 Ajax support does not rely on any third-party libraries and properly namespaces itself using the OpenAjax Alliance Hub, ensuring that no collisions occur with other Javascript libraries.



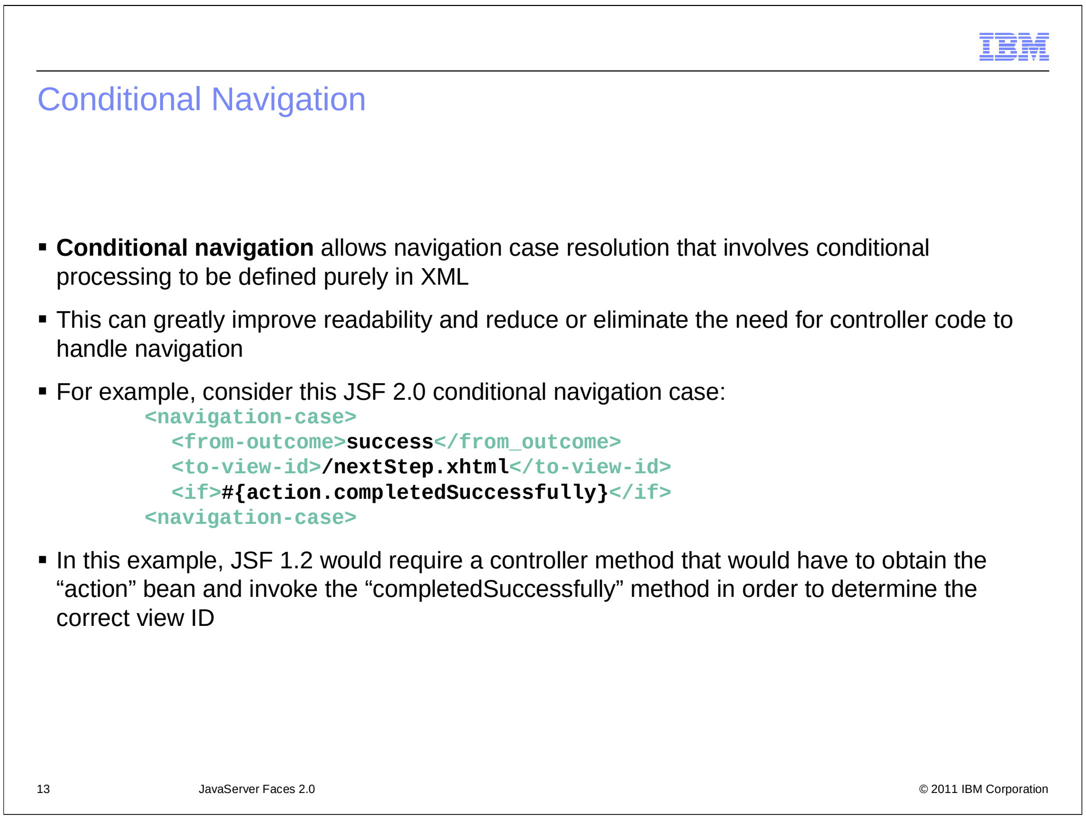
Implicit navigation

- **Implicit navigation** is used as a “fall back” mechanism in navigation case resolution. That is, if an outcome does not have a corresponding view ID defined, a default view ID is chosen based on the outcome name.
- For example, if an outcome of “success” is specified but there is no navigation case defined for that outcome, the view ID “/success.xhtml” is used automatically.
- This greatly reduces the amount of boiler plate code required in faces-config.xml since most applications have a 1:1 mapping between outcome names and view IDs

12 JavaServer Faces 2.0 © 2011 IBM Corporation

Implicit navigation is a JSF 2.0 feature that greatly reduces the amount of boilerplate configuration required in faces-config.xml. JSF navigation is based on having a named outcome be derived from some action, such as clicking a button. This named outcome must declare which page it corresponds with in faces-config.xml. Typically there is a very simple one-to-one correspondence between outcome names and the pages they refer to. Namely, an outcome with name “success” corresponds to the page - or view ID - “/success.xhtml”.

Declaring these relationships in faces-config.xml is tedious and complicates the descriptor. Fortunately, JSF 2.0 will automatically assume the previous relationship between outcome and page names, meaning that developers can refer to a page named “/page.xhtml” with the outcome name “page” and not have to declare that mapping in faces-config.xml.



Conditional Navigation

- **Conditional navigation** allows navigation case resolution that involves conditional processing to be defined purely in XML
- This can greatly improve readability and reduce or eliminate the need for controller code to handle navigation
- For example, consider this JSF 2.0 conditional navigation case:

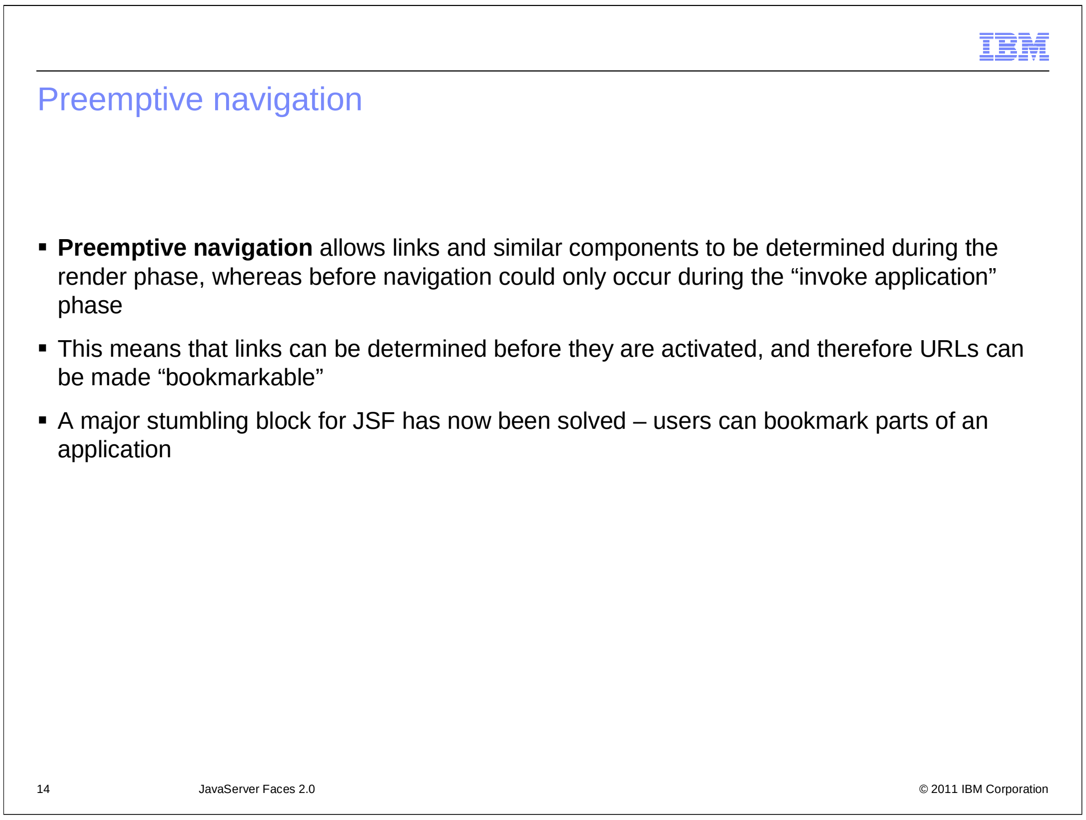
```
<navigation-case>
  <from-outcome>success</from_outcome>
  <to-view-id>/nextStep.xhtml</to-view-id>
  <if>#{action.completedSuccessfully}</if>
<navigation-case>
```

- In this example, JSF 1.2 would require a controller method that would have to obtain the “action” bean and invoke the “completedSuccessfully” method in order to determine the correct view ID

13 JavaServer Faces 2.0 © 2011 IBM Corporation

Before JSF 2.0, conditional navigation always required a backend controller method to invoke code in order to determine the correct outcome for the given inputs. Most cases consist of checking a few Boolean bean methods, meaning that a large amount of boilerplate code is required on the backend. JSF 2.0 introduces a simplified form of conditional navigation whereby simple cases can be specified using EL expressions in the faces-config.xml descriptor. For most conditional navigation cases, this capability greatly reduces the amount of code needed on the backend.

Consider the example provided. With JSF 2.0, this conditional navigation – which is very typical of JSF applications – can be easily expressed with the faces-config.xml descriptor. An older JSF application requires a non-trivial amount of boilerplate code used to obtain a bean and check a method invocation.

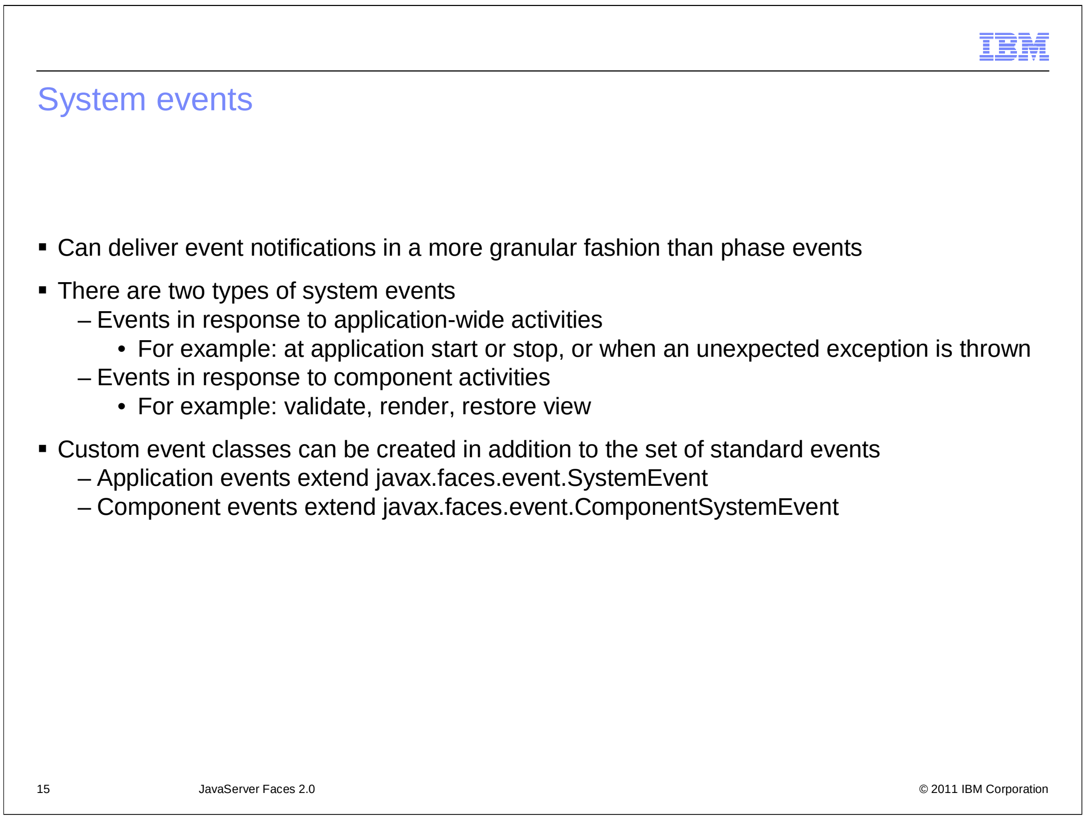


Preemptive navigation

- **Preemptive navigation** allows links and similar components to be determined during the render phase, whereas before navigation could only occur during the “invoke application” phase
- This means that links can be determined before they are activated, and therefore URLs can be made “bookmarkable”
- A major stumbling block for JSF has now been solved – users can bookmark parts of an application

14 JavaServer Faces 2.0 © 2011 IBM Corporation

A common complaint of JSF is its inability to generate bookmarkable links due to the navigation model, which requires backend code to be invoked before navigation occurs. JSF 2.0 addresses this problem by allowing links (and similar components) to be resolved during the render phase of the JSF life cycle, finally allowing for bookmarkable URLs.

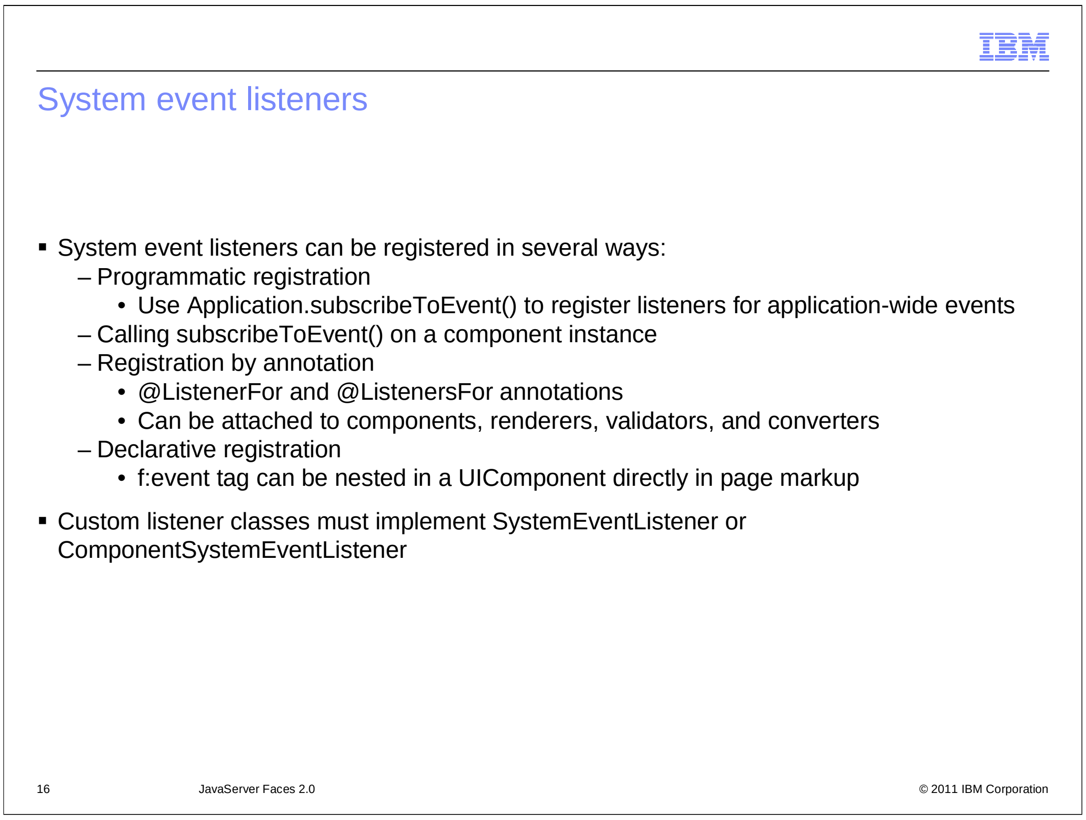


System events

- Can deliver event notifications in a more granular fashion than phase events
- There are two types of system events
  - Events in response to application-wide activities
    - For example: at application start or stop, or when an unexpected exception is thrown
  - Events in response to component activities
    - For example: validate, render, restore view
- Custom event classes can be created in addition to the set of standard events
  - Application events extend javax.faces.event.SystemEvent
  - Component events extend javax.faces.event.ComponentSystemEvent

15 JavaServer Faces 2.0 © 2011 IBM Corporation

Before JSF 2.0, the ability to fire off arbitrary events could only be achieved during the execution of JSF life cycle phases. In JSF 2.0, system events can be assigned in a much more fine-grained fashion. For example, system events can be fired in response to application-specific activities (such as starting or stopping, or when some unexpected exception is thrown) or during component activities (such as during validation or rendering). There are several standard system events and developers can create custom ones as well.

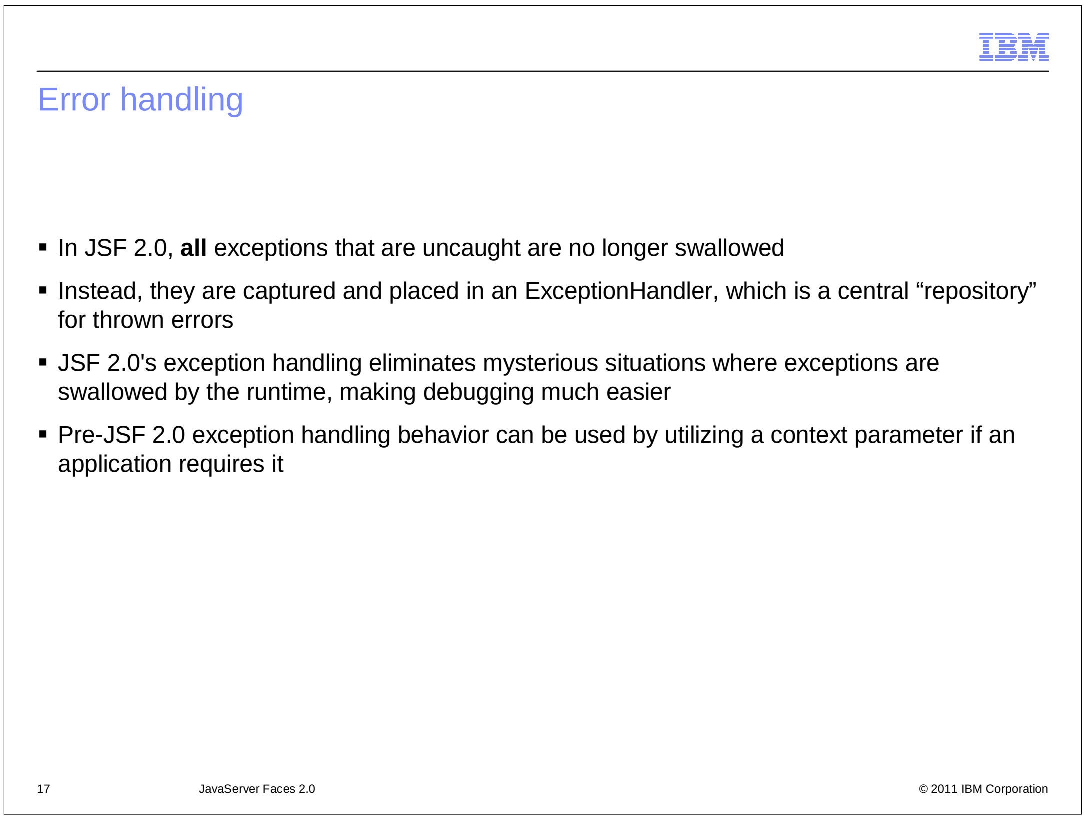


System event listeners

- System event listeners can be registered in several ways:
  - Programmatic registration
    - Use Application.subscribeToEvent() to register listeners for application-wide events
  - Calling subscribeToEvent() on a component instance
  - Registration by annotation
    - @ListenerFor and @ListenersFor annotations
    - Can be attached to components, renderers, validators, and converters
  - Declarative registration
    - f:event tag can be nested in a UIComponent directly in page markup
- Custom listener classes must implement SystemEventListener or ComponentSystemEventListener

16 JavaServer Faces 2.0 © 2011 IBM Corporation

Along with being able to fire off system events, JSF 2.0 allows developers to create system event listeners. These listeners can be registered in a variety of ways, either programmatically (for application-, system-, and component-wide events), using an annotation (for components, renderers, validators, and converters) or declaratively (for components).



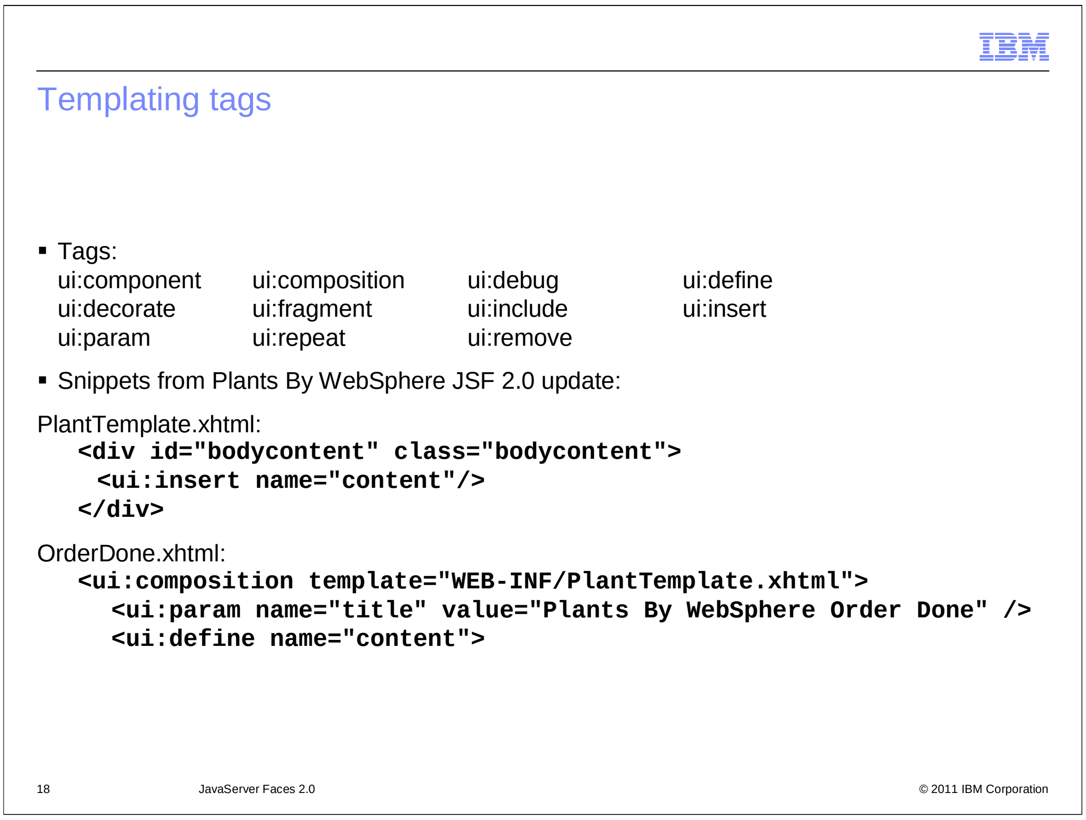
Error handling

- In JSF 2.0, **all** exceptions that are uncaught are no longer swallowed
- Instead, they are captured and placed in an ExceptionHandler, which is a central “repository” for thrown errors
- JSF 2.0's exception handling eliminates mysterious situations where exceptions are swallowed by the runtime, making debugging much easier
- Pre-JSF 2.0 exception handling behavior can be used by utilizing a context parameter if an application requires it

17 JavaServer Faces 2.0 © 2011 IBM Corporation

Before JSF 2.0, unexpected exceptions (that is, those that do not conform to exceptions thrown by standard JSF life cycle APIs) are swallowed and logged. In JSF 2.0, this is no longer the case with the ExceptionHandler exception repository. All unhandled exceptions are captured by the ExceptionHandler and reported later. This helps developers debug applications where previous version of JSF would result in mysterious situations where the application does not work yet no exception is presented to the user.

If the pre-JSF 2.0 behavior is required, a context parameter can be specified to restore the pre-JSF 2.0 exception handling facilities.



Templating tags

- Tags:

ui:component	ui:composition	ui:debug	ui:define
ui:decorate	ui:fragment	ui:include	ui:insert
ui:param	ui:repeat	ui:remove	

- Snippets from Plants By WebSphere JSF 2.0 update:

PlantTemplate.xhtml:

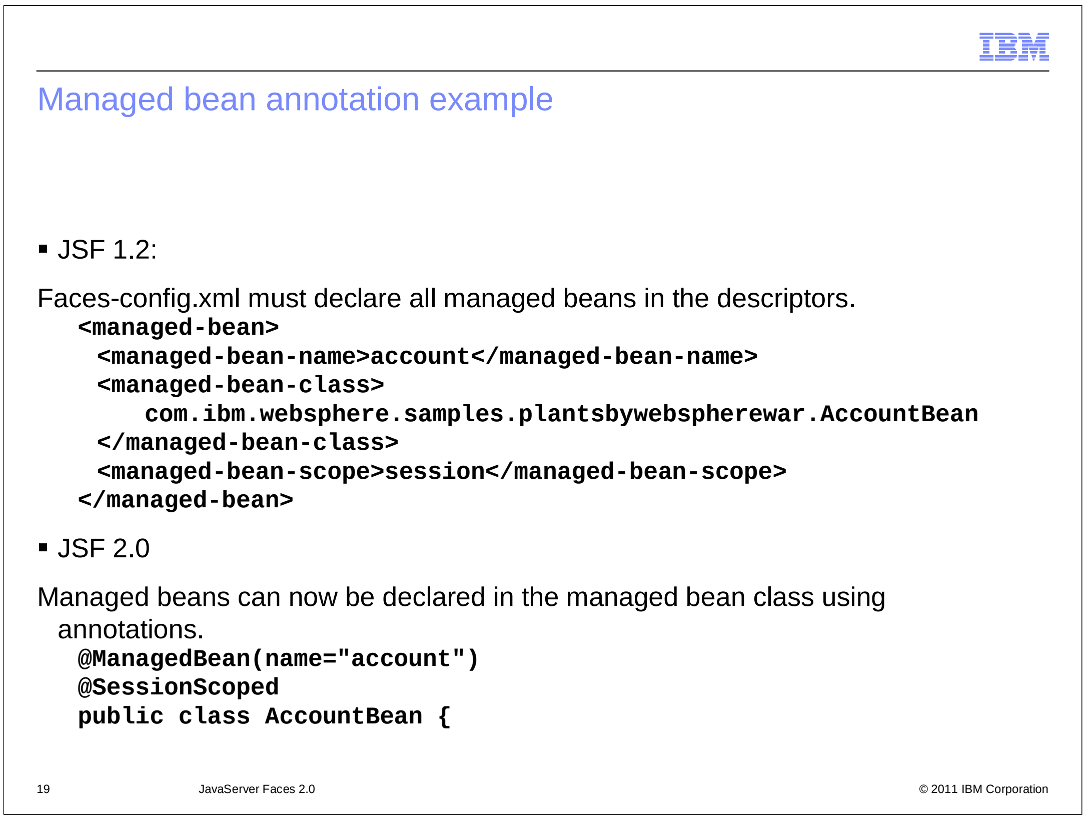
```
<div id="bodycontent" class="bodycontent">
  <ui:insert name="content"/>
</div>
```

OrderDone.xhtml:

```
<ui:composition template="WEB-INF/PlantTemplate.xhtml">
   <ui:param name="title" value="Plants By WebSphere Order Done" />
   <ui:define name="content">
```

18 JavaServer Faces 2.0 © 2011 IBM Corporation

This example illustrates some of the new JSF 2.0 templating tags. A <ui:composition> tag is used to define a composition, or fragment, to be reused. The fragment is another XHTML page which can be located in WEB-INF for security. Parameters that can alter the template are specified with <ui:param> and the name of the composition is defined by <ui:define>. To insert the composition, the <ui:insert> tag is used.



Managed bean annotation example

- JSF 1.2:

Faces-config.xml must declare all managed beans in the descriptors.

```
<managed-bean>
  <managed-bean-name>account</managed-bean-name>
  <managed-bean-class>
      com.ibm.websphere.samples.plantsbywebspherewar.AccountBean
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

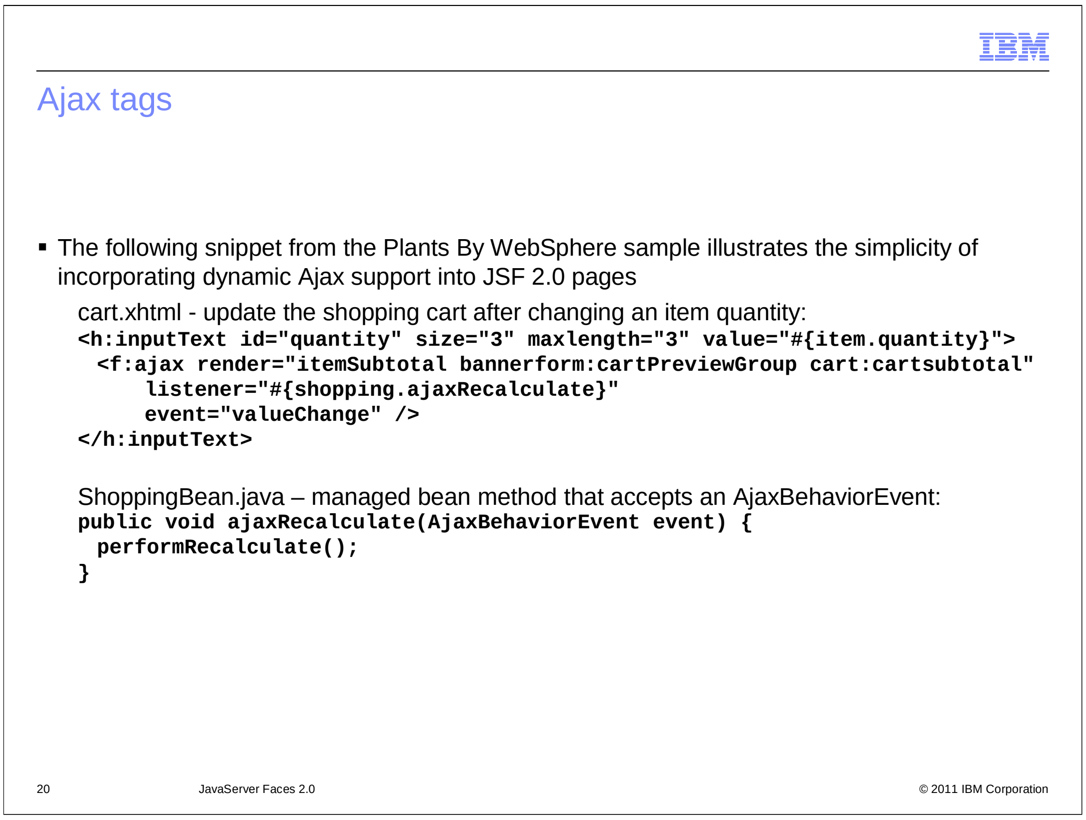
- JSF 2.0

Managed beans can now be declared in the managed bean class using annotations.

```
@ManagedBean(name="account")
@SessionScoped
public class AccountBean {
```

19 JavaServer Faces 2.0 © 2011 IBM Corporation

In JSF 1.2, defining a managed bean takes a large amount of markup in faces-config.xml. Using the new JSF 2.0 annotations, the same bean can be specified by using the @ManagedBean annotation along with the @SessionScoped annotation (which declares the bean as having session scope).



Ajax tags

- The following snippet from the Plants By WebSphere sample illustrates the simplicity of incorporating dynamic Ajax support into JSF 2.0 pages

  cart.xhtml - update the shopping cart after changing an item quantity:

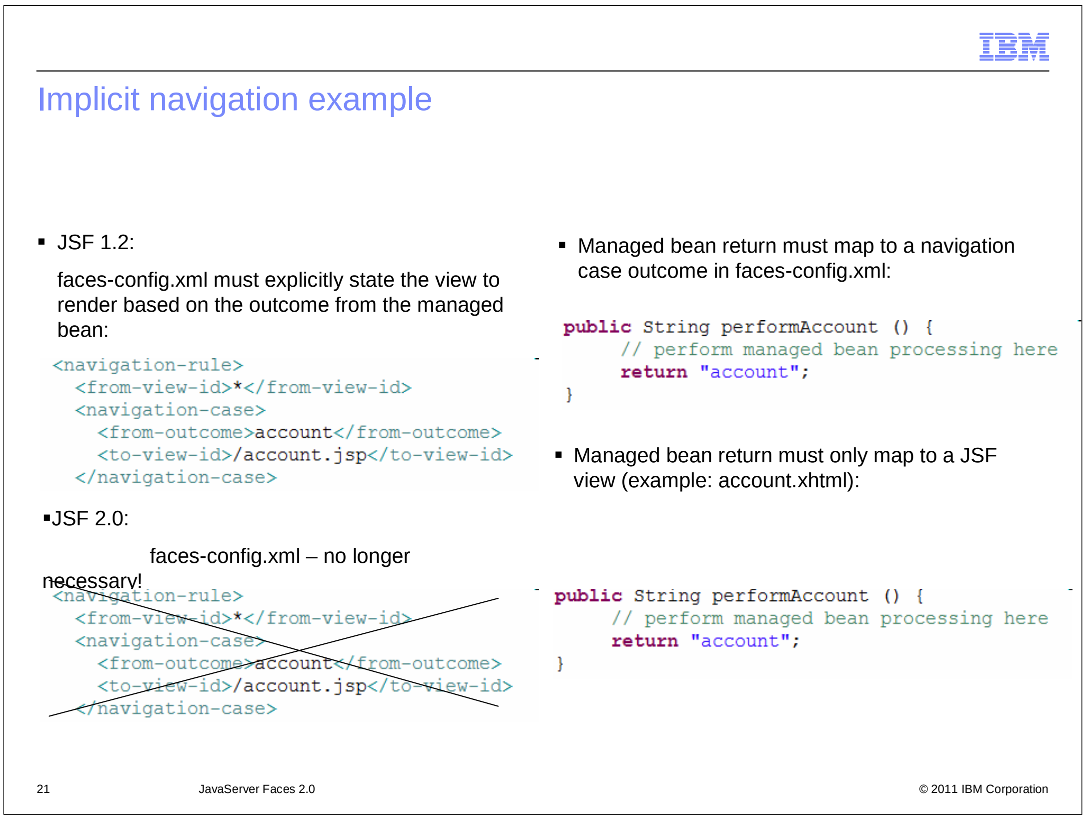
```
<h:inputText id="quantity" size="3" maxlength="3" value="#{item.quantity}">
  <f:ajax render="itemSubtotal bannerform:cartPreviewGroup cart:cartsubtotal"
      listener="#{shopping.ajaxRecalculate}"
      event="valueChange" />
</h:inputText>
```

  ShoppingBean.java – managed bean method that accepts an AjaxBehaviorEvent:

```
public void ajaxRecalculate(AjaxBehaviorEvent event) {
  performRecalculate();
}
```

20 JavaServer Faces 2.0 © 2011 IBM Corporation

The <f:ajax> tag is used to enable Ajax support for JSF components. The tag has many different uses and semantics depending on how it is used, but the most common case is illustrated by this example. Here you have an input component that fires an Ajax event upon a value change (“valueChange”) event. The listener attribute specifies the bean method to invoke and the render attribute specifies which components are to be re-rendered after the Ajax event has fired and a result (if any) has been returned.



Implicit navigation example

- JSF 1.2:

  faces-config.xml must explicitly state the view to render based on the outcome from the managed bean:

```
<navigation-rule>
  <from-view-id>*</from-view-id>
  <navigation-case>
    <from-outcome>account</from-outcome>
    <to-view-id>/account.jsp</to-view-id>
  </navigation-case>
```

- Managed bean return must map to a navigation case outcome in faces-config.xml:

```
public String performAccount () {
     // perform managed bean processing here
     return "account";
}
```

- JSF 2.0:

  faces-config.xml – no longer necessary!

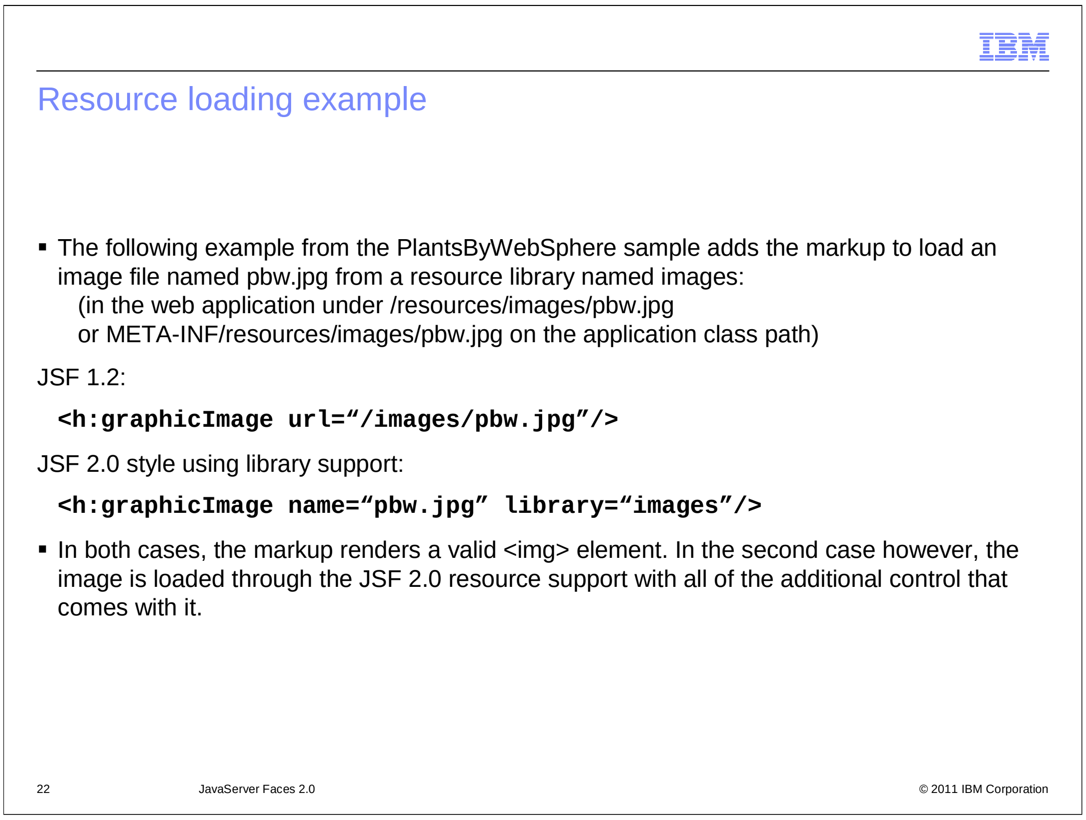
```
~~<navigation-rule>~~
  ~~<from-view-id>*</from-view-id>~~
  ~~<navigation-case>~~
    ~~<from-outcome>account</from-outcome>~~
    ~~<to-view-id>/account.jsp</to-view-id>~~
  ~~</navigation-case>~~
```

- Managed bean return must only map to a JSF view (example: account.xhtml):

```
public String performAccount () {
     // perform managed bean processing here
     return "account";
}
```

21 JavaServer Faces 2.0 © 2011 IBM Corporation

This example illustrates the ease with which most common navigation cases are handled through the new JSF 2.0 implicit navigation feature. In this example, the performAccount() method acts as a controller method which returns an outcome named “account”. Before JSF 2.0, the developer had to map the “account” outcome to the page (view ID) “/account.jsp”. With implicit navigation, the JSF runtime will automatically attempt to locate a view ID with an equivalently-mapped named (in this case, “/account.xhtml”).



Resource loading example

- The following example from the PlantsByWebSphere sample adds the markup to load an image file named pbw.jpg from a resource library named images:
  (in the web application under /resources/images/pbw.jpg
  or META-INF/resources/images/pbw.jpg on the application class path)

JSF 1.2:

```
<h:graphicImage url="/images/pbw.jpg"/>
```

JSF 2.0 style using library support:

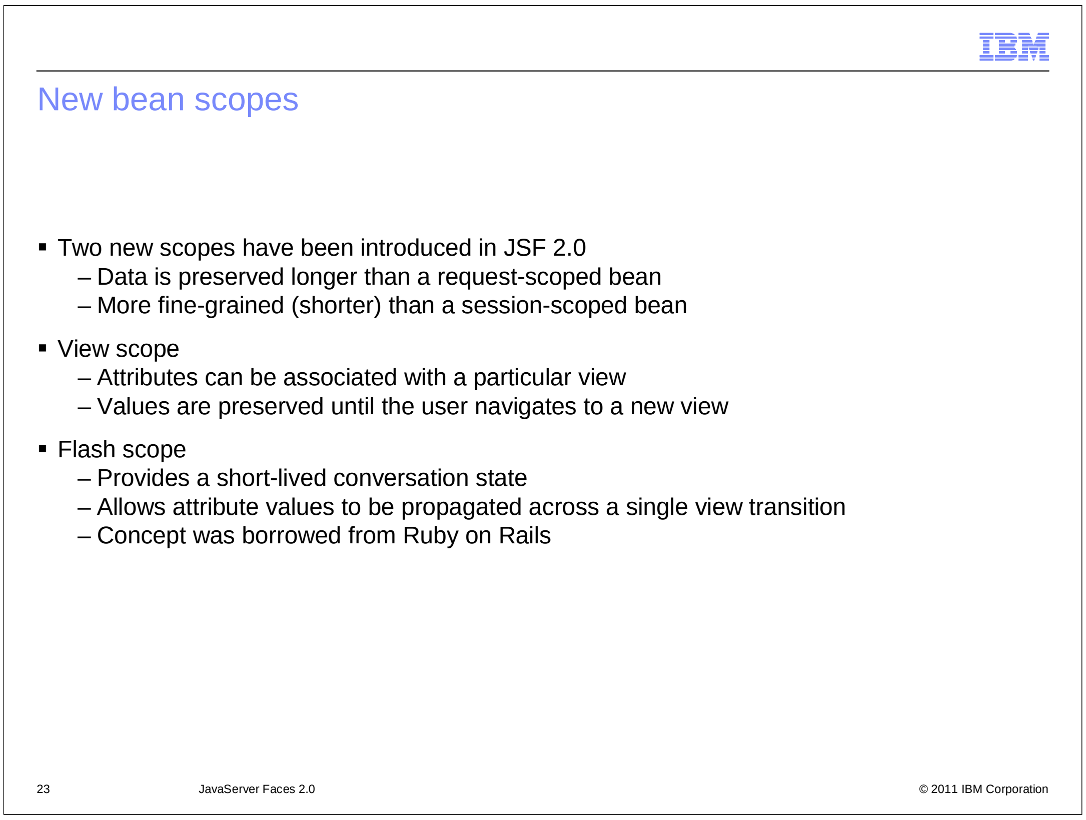
```
<h:graphicImage name="pbw.jpg" library="images"/>
```

- In both cases, the markup renders a valid <img> element. In the second case however, the image is loaded through the JSF 2.0 resource support with all of the additional control that comes with it.

22 JavaServer Faces 2.0 © 2011 IBM Corporation

Before JSF 1.2, all aspects of resource loading had to be handled by the application developer. Typically this meant that either a naming convention had to be devised for naming and version handling of resources or a custom Servlet had to be used to make resources residing in a protected location accessible. In JSF 2.0, resources can be referenced through attributes specifying a name, library, and version. These attributes are automatically resolved to either an application directory in the form "/resources/<library>/<version>/<name>" or a protected directory "/META-INF/resources/<library>/<version>/<name>".

This example shows the difference between both approaches. Before JSF 2.0, resources had to be referenced by URL. In JSF 2.0, resources can be referenced using name, library, and version.

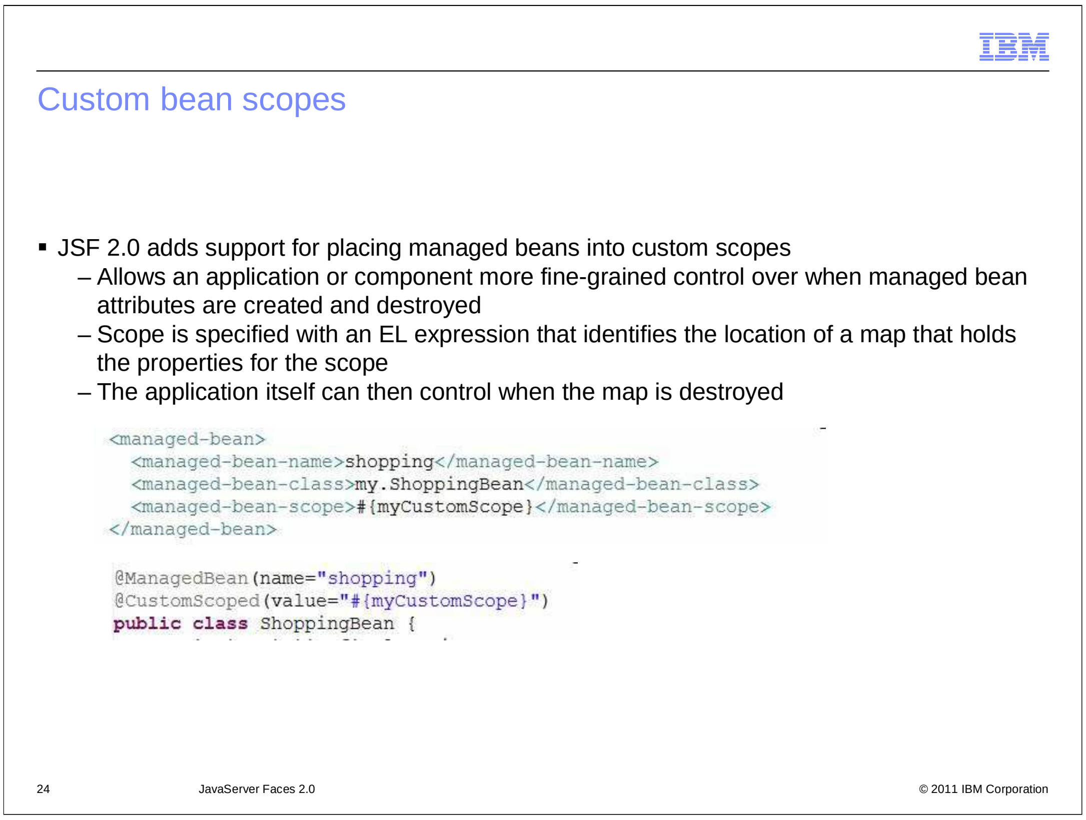


New bean scopes

- Two new scopes have been introduced in JSF 2.0
  - Data is preserved longer than a request-scoped bean
  - More fine-grained (shorter) than a session-scoped bean
- View scope
  - Attributes can be associated with a particular view
  - Values are preserved until the user navigates to a new view
- Flash scope
  - Provides a short-lived conversation state
  - Allows attribute values to be propagated across a single view transition
  - Concept was borrowed from Ruby on Rails

23 JavaServer Faces 2.0 © 2011 IBM Corporation

There are two new bean scopes available in JSF 2.0: view and flash. These new scopes allow data to be preserved for a duration longer than a request but shorter than a session. The view scope allows attributes to be associated with a view, meaning that the values are preserved until the user navigates to a new view. The flash scope mimics a short-lived conversation state. Attributes are preserved between a single view transition, from one page to another but no further. This scope incorporates ideas borrowed from the popular Ruby on Rails framework.



Custom bean scopes

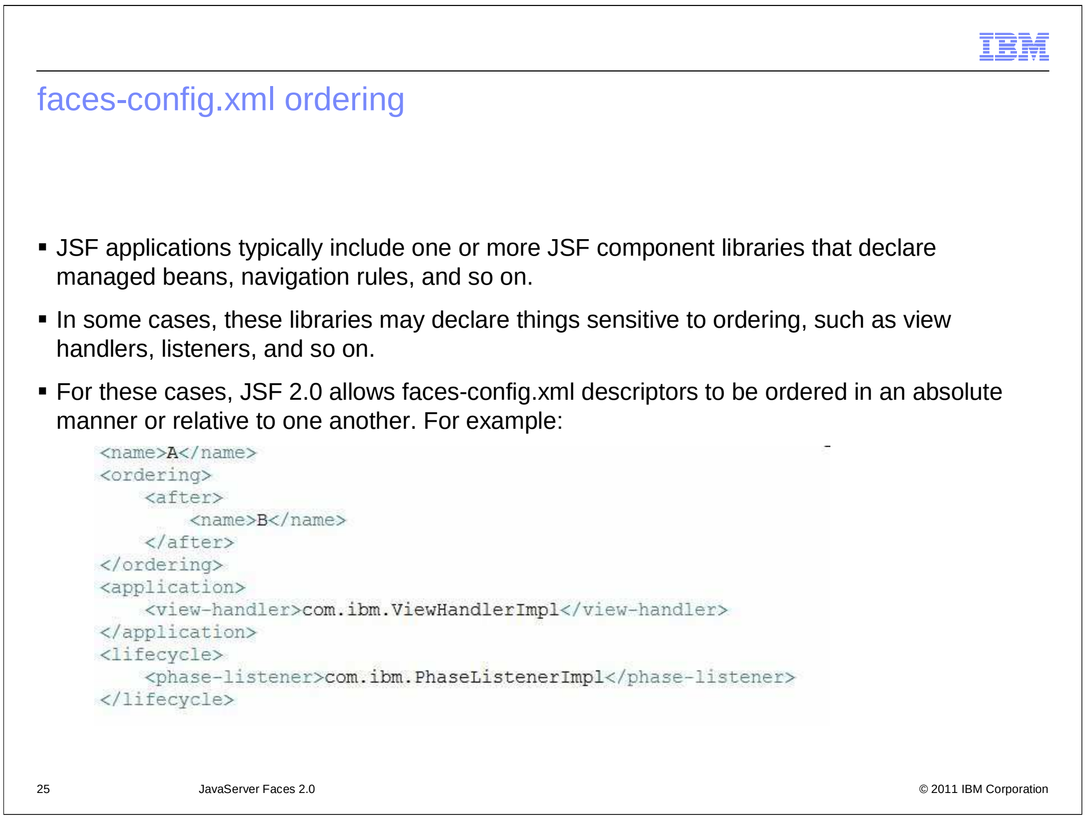
- JSF 2.0 adds support for placing managed beans into custom scopes
  - Allows an application or component more fine-grained control over when managed bean attributes are created and destroyed
  - Scope is specified with an EL expression that identifies the location of a map that holds the properties for the scope
  - The application itself can then control when the map is destroyed

```
<managed-bean>
  <managed-bean-name>shopping</managed-bean-name>
  <managed-bean-class>my.ShoppingBean</managed-bean-class>
  <managed-bean-scope>#{myCustomScope}</managed-bean-scope>
</managed-bean>
```

```
@ManagedBean(name="shopping")
@CustomScoped(value="#{myCustomScope}")
public class ShoppingBean {
```

24 JavaServer Faces 2.0 © 2011 IBM Corporation

In addition to the standard bean scopes, developers can create custom bean scopes with JSF 2.0 for situations where a very specific attribute lifespan is required. As seen in the example, developers can declare a custom scope in faces-config.xml or by using the @CustomScoped annotation. An EL expression is used to identify the map which holds the scope attributes. The application can control the creation and destruction of this map.



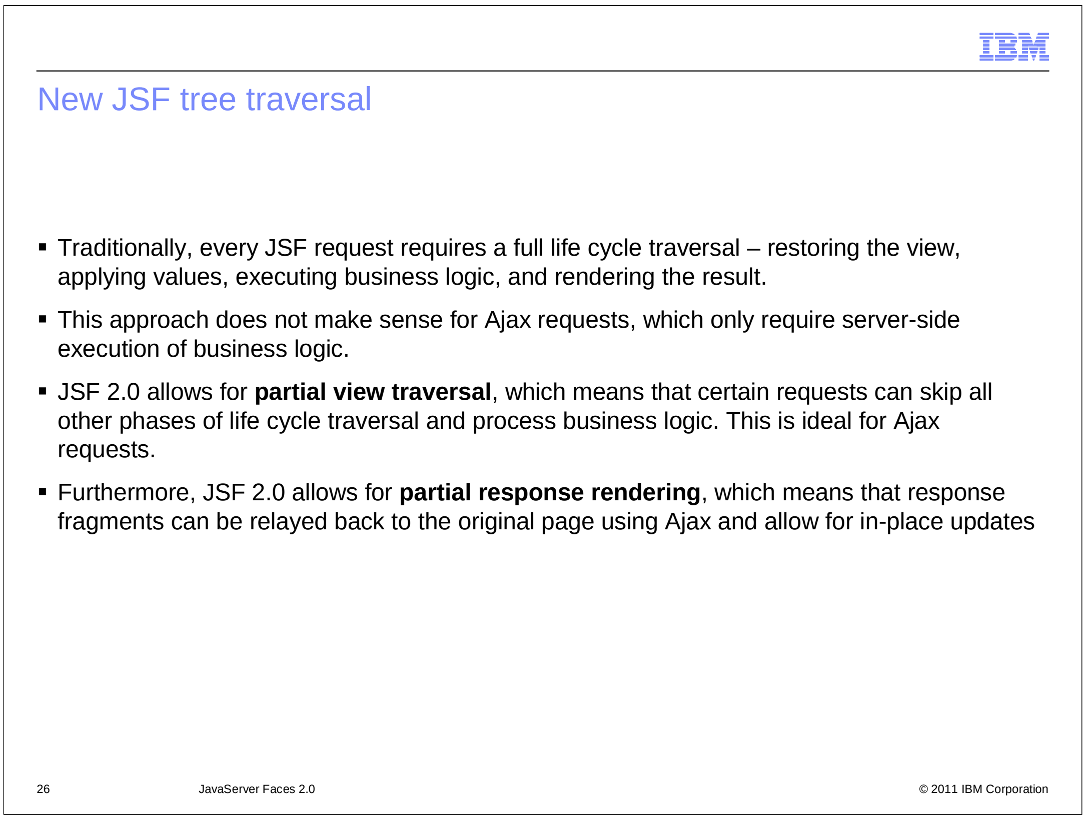
faces-config.xml ordering

- JSF applications typically include one or more JSF component libraries that declare managed beans, navigation rules, and so on.
- In some cases, these libraries may declare things sensitive to ordering, such as view handlers, listeners, and so on.
- For these cases, JSF 2.0 allows faces-config.xml descriptors to be ordered in an absolute manner or relative to one another. For example:

```
<name>A</name>
<ordering>
    <after>
        <name>B</name>
    </after>
</ordering>
<application>
    <view-handler>com.ibm.ViewHandlerImpl</view-handler>
</application>
<lifecycle>
    <phase-listener>com.ibm.PhaseListenerImpl</phase-listener>
</lifecycle>
```

25 JavaServer Faces 2.0 © 2011 IBM Corporation

Typical JSF applications generally consist of one or more third-party libraries that may declare JSF components, life cycle listeners, and so on. Before JSF 2.0, the ordering of such declarations was generally indeterminate, which could lead to issues, particularly when an expected ordering for listeners was expected. In JSF 2.0, each faces-config.xml descriptor can be named and an <ordering> element can be specified to dictate where the descriptor should fall (via a relative or absolute ordering) in the overall order of application descriptors.

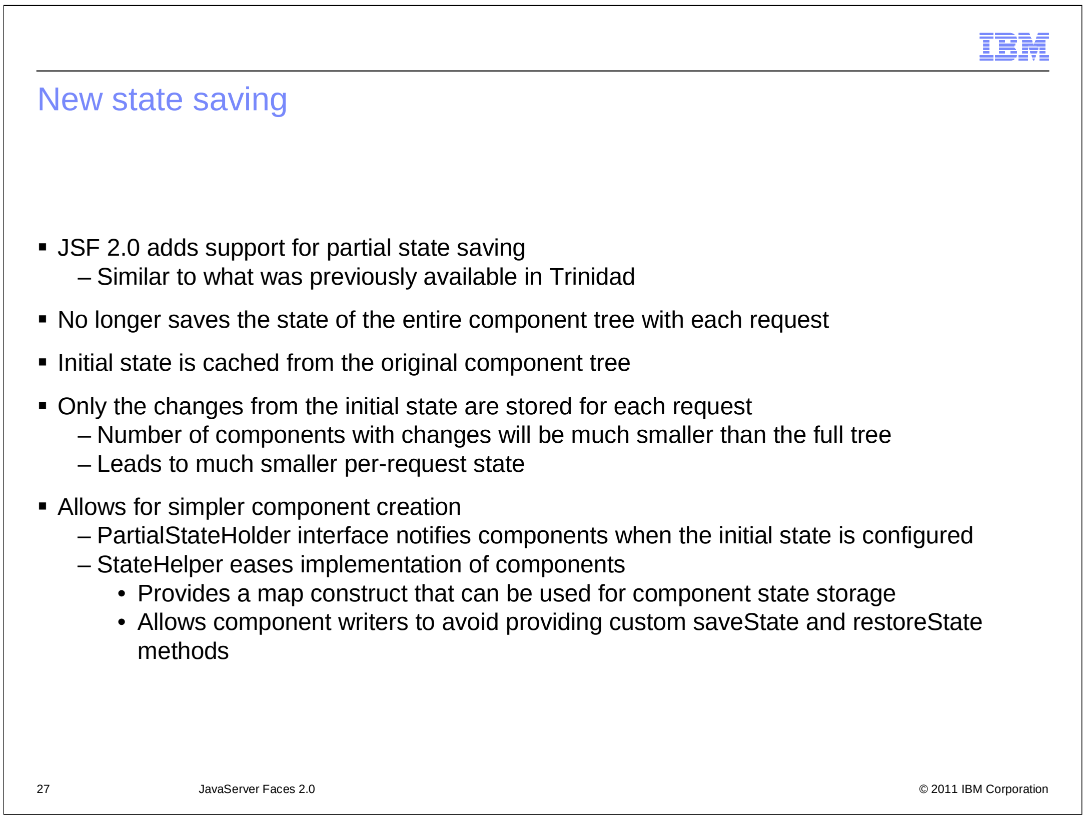


New JSF tree traversal

- Traditionally, every JSF request requires a full life cycle traversal – restoring the view, applying values, executing business logic, and rendering the result.
- This approach does not make sense for Ajax requests, which only require server-side execution of business logic.
- JSF 2.0 allows for **partial view traversal**, which means that certain requests can skip all other phases of life cycle traversal and process business logic. This is ideal for Ajax requests.
- Furthermore, JSF 2.0 allows for **partial response rendering**, which means that response fragments can be relayed back to the original page using Ajax and allow for in-place updates

26 JavaServer Faces 2.0 © 2011 IBM Corporation

Before JSF 2.0, every request would require a full traversal through the JSF life cycle: restoring the view, applying values, executing business logic, and rendering the result. With the introduction of Ajax support in JSF 2.0, this approach does not make sense as Ajax requests only need to process business logic and render a result; this feature is called partial view traversal. An additional capability tailored to the introduction of Ajax support is partial response rendering, which means that only the fragments of the page which changed as a result of an Ajax request are sent back to the page. Once received, the page is updated in-place.



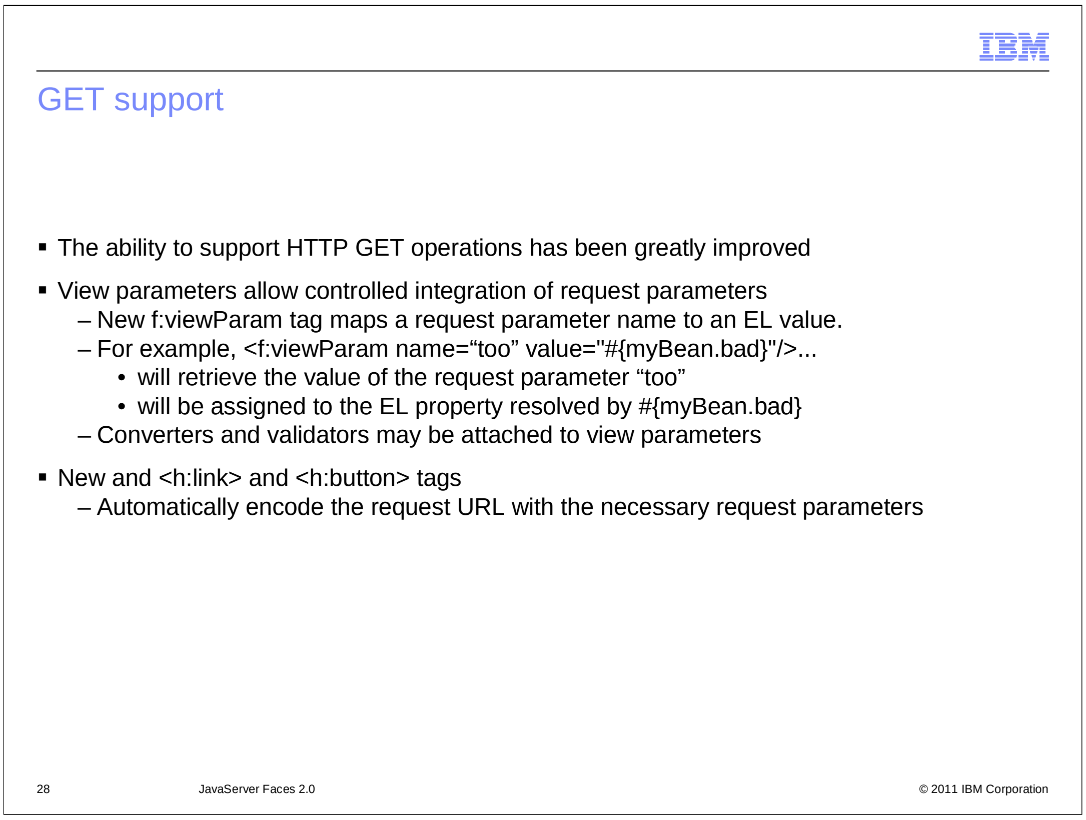
New state saving

- JSF 2.0 adds support for partial state saving
  - Similar to what was previously available in Trinidad
- No longer saves the state of the entire component tree with each request
- Initial state is cached from the original component tree
- Only the changes from the initial state are stored for each request
  - Number of components with changes will be much smaller than the full tree
  - Leads to much smaller per-request state
- Allows for simpler component creation
  - PartialStateHolder interface notifies components when the initial state is configured
  - StateHelper eases implementation of components
    - Provides a map construct that can be used for component state storage
    - Allows component writers to avoid providing custom saveState and restoreState methods

27 JavaServer Faces 2.0 © 2011 IBM Corporation

Before JSF 2.0, every request/response cycle would require the entire component tree to be saved. In typical JSF applications, this would generally result in a great deal of memory consumption. JSF 2.0 now allows for partial state saving, a feature similar to the one available for the Trinidad widget library. With partial state saving, only the initial component tree state is saved and updates are saved as deltas, leading to a much smaller per-request state.

JSF 2.0 introduces new APIs to enable component developers to easily use this feature. By implementing the PartialStateHolder interface, components are notified when the initial state is configured and the StateHelper class can be used to avoid having to write custom saveState() and restoreState() methods.



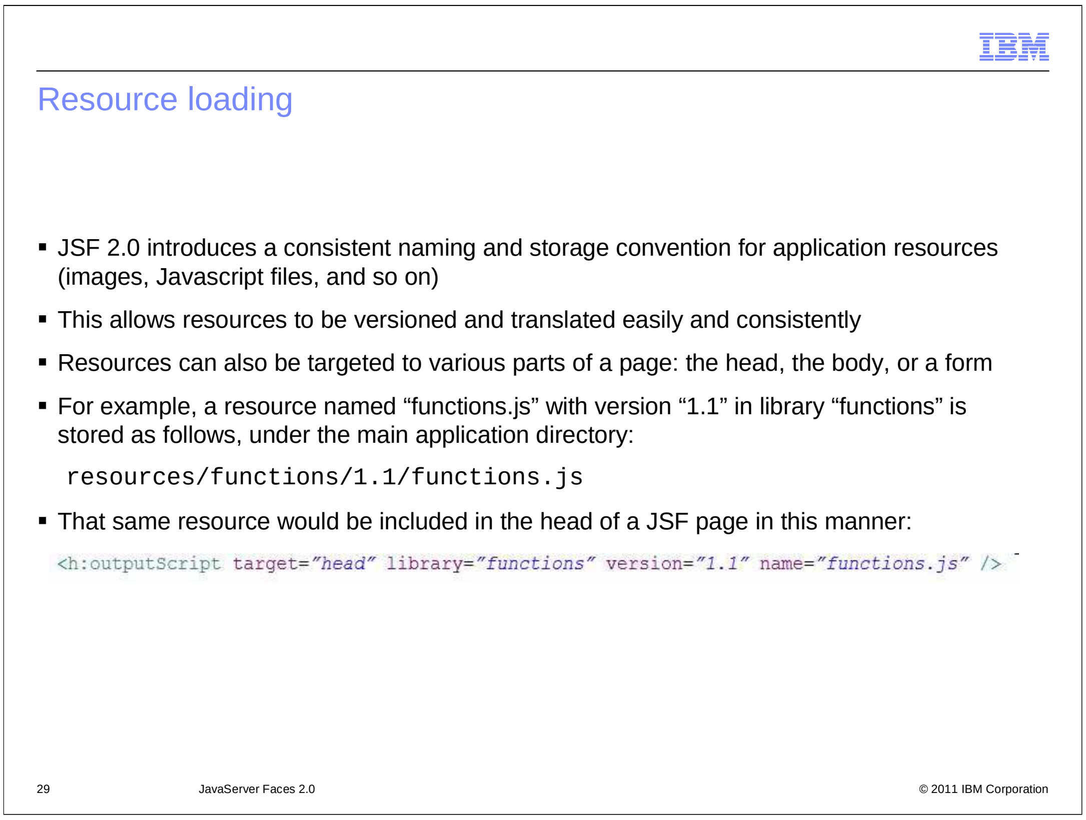
GET support

- The ability to support HTTP GET operations has been greatly improved
- View parameters allow controlled integration of request parameters
  - New f:viewParam tag maps a request parameter name to an EL value.
  - For example, <f:viewParam name="too" value="#{myBean.bad}"/>...
    - will retrieve the value of the request parameter "too"
    - will be assigned to the EL property resolved by #{myBean.bad}
  - Converters and validators may be attached to view parameters
- New and <h:link> and <h:button> tags
  - Automatically encode the request URL with the necessary request parameters

28 JavaServer Faces 2.0 © 2011 IBM Corporation

Traditionally, HTTP GET support with JSF has been poor as the primary method of request submission has always been PUT. With JSF 2.0, support for GET operations has been greatly improved. With the <f:viewParam> tag, developers can take a request parameter (specified by the name attribute) and assign it to an EL property (specified by the value attribute). And, like any other JSF component, converters and validators can be assigned to these view parameters.

Additionally, JSF 2.0 introduces the <h:link> and <h:button> tags, which will automatically encode the request URL with the necessary request parameters.



Resource loading

- JSF 2.0 introduces a consistent naming and storage convention for application resources (images, Javascript files, and so on)
- This allows resources to be versioned and translated easily and consistently
- Resources can also be targeted to various parts of a page: the head, the body, or a form
- For example, a resource named “functions.js” with version “1.1” in library “functions” is stored as follows, under the main application directory:

```
resources/functions/1.1/functions.js
```

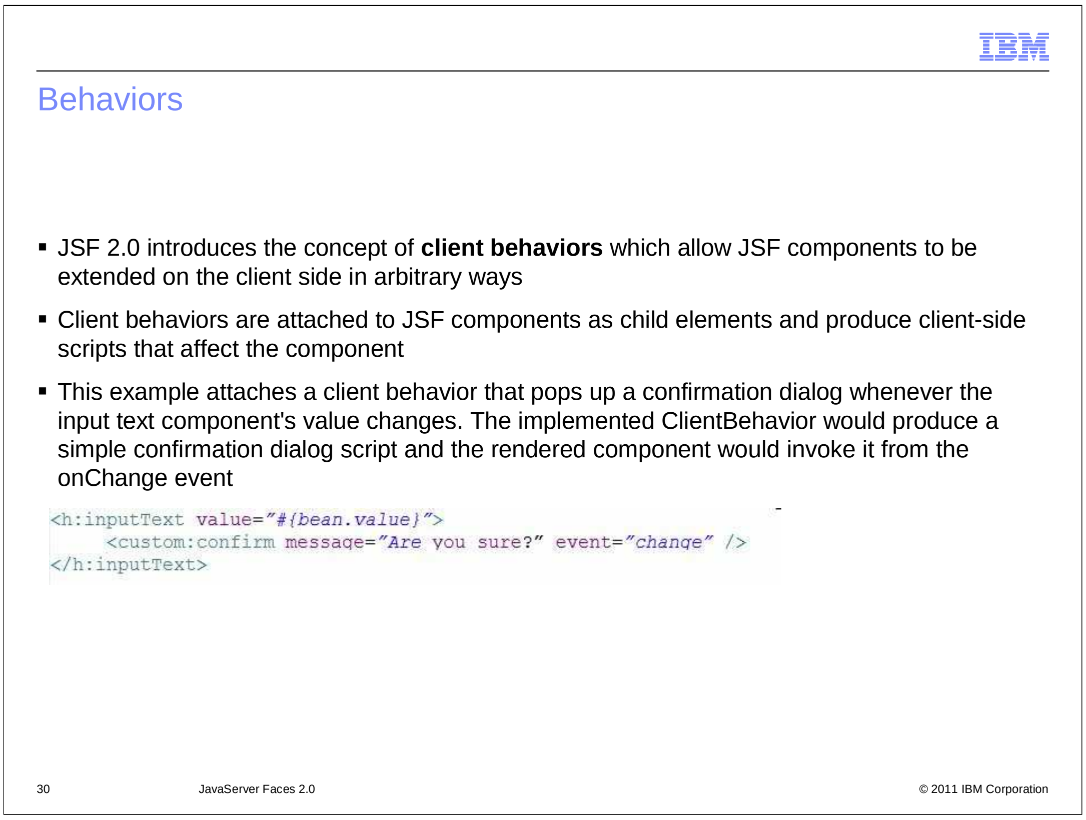
- That same resource would be included in the head of a JSF page in this manner:

```
<h:outputScript target="head" library="functions" version="1.1" name="functions.js" />
```

29 JavaServer Faces 2.0 © 2011 IBM Corporation

Before JSF 2.0, application developers had to devise their own schemes for resource handling. With JSF 2.0’s integrated support for resources, developers can now use a robust system that supports resource libraries and version handling. Any tag which can use a resource (such as a script tag, image tag, and so on) can emit a resource specified either directly by a URL (as would be the case before JSF 2.0) or by a combination of resource name, library, and version. Resources are then automatically translated either to the application directory “resources” or the protected application directory “META-INF/resources”.

In the example above, a resource is specified with the name “functions.js”, library “functions”, and version “1.1”. The rendered resource URL is determined to be “resources/functions/1.1./functions.js” because, in this particular case, the resource appears within the application “resources” directory. However, if it were located in the protected application directory “META-INF/resources”, a different URL would have been rendered.



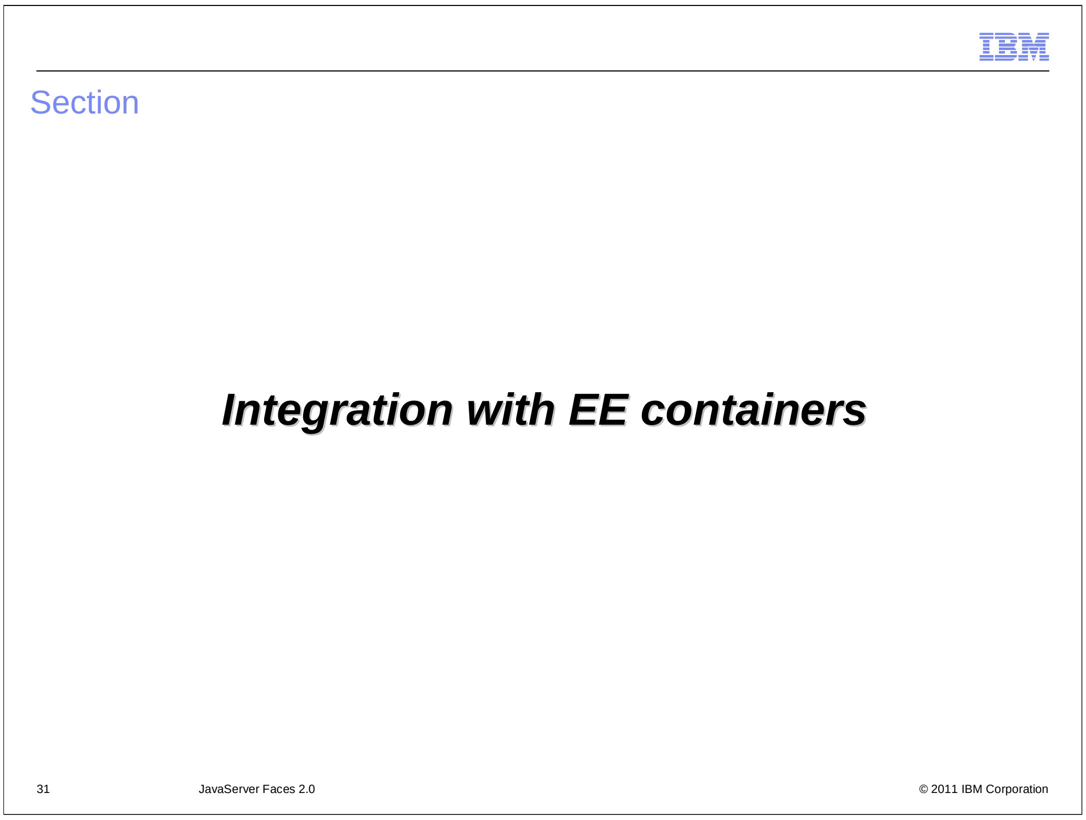
Behaviors

- JSF 2.0 introduces the concept of **client behaviors** which allow JSF components to be extended on the client side in arbitrary ways
- Client behaviors are attached to JSF components as child elements and produce client-side scripts that affect the component
- This example attaches a client behavior that pops up a confirmation dialog whenever the input text component's value changes. The implemented ClientBehavior would produce a simple confirmation dialog script and the rendered component would invoke it from the onChange event

```
<h:inputText value="#{bean.value}">
     <custom:confirm message="Are you sure?" event="change" />
</h:inputText>
```

In order to support Ajax within JSF 2.0, the concept of client behaviors was introduced. Client behaviors allow JSF components to interact with the client side in arbitrary ways. Developers attach client behaviors to JSF components as child elements and the client behaviors will render Javascript used to interact with the client side.

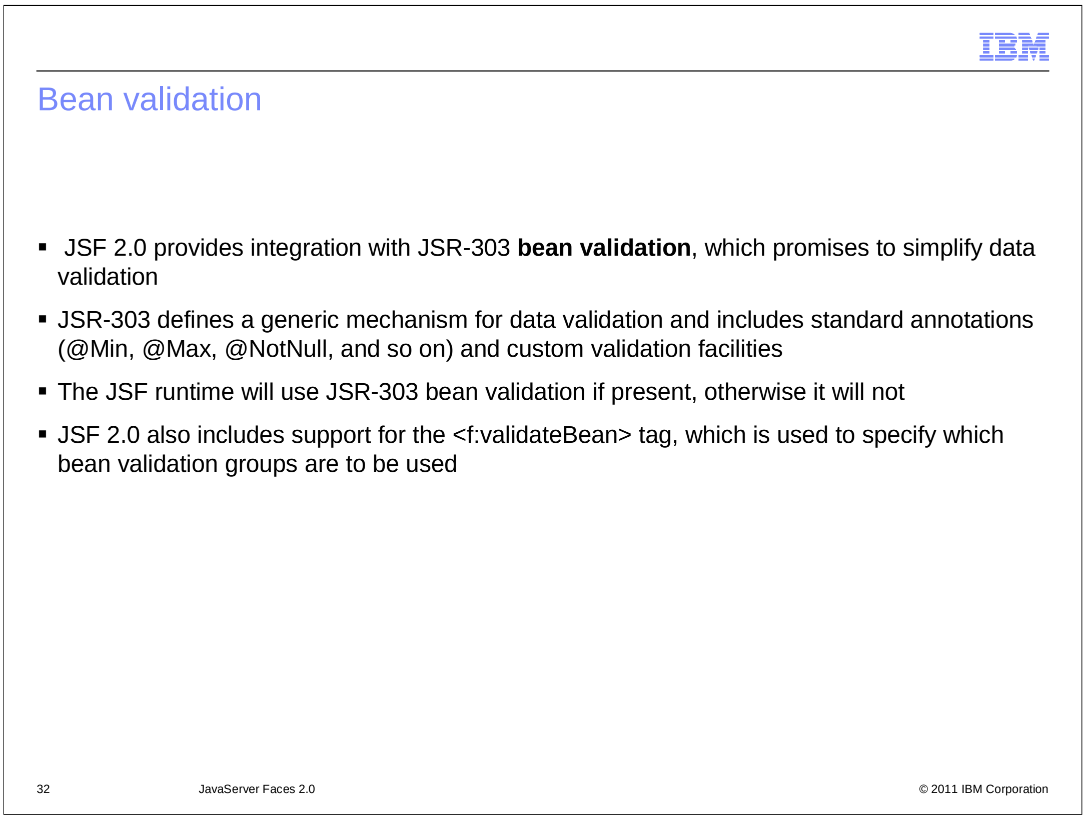
In the provided example, an input component has a client behavior attached to it which will pop up a dialog box whenever the “change” event occurs. Though not shown, the rendered Javascript consists of an alert() method call. The JSF runtime renders the code necessary to attach the custom component code to the “change” event.



Section

Integration with EE containers

JSF 2.0 integrates with some new Java EE 6 containers. This section will show which containers and how they are integrated.



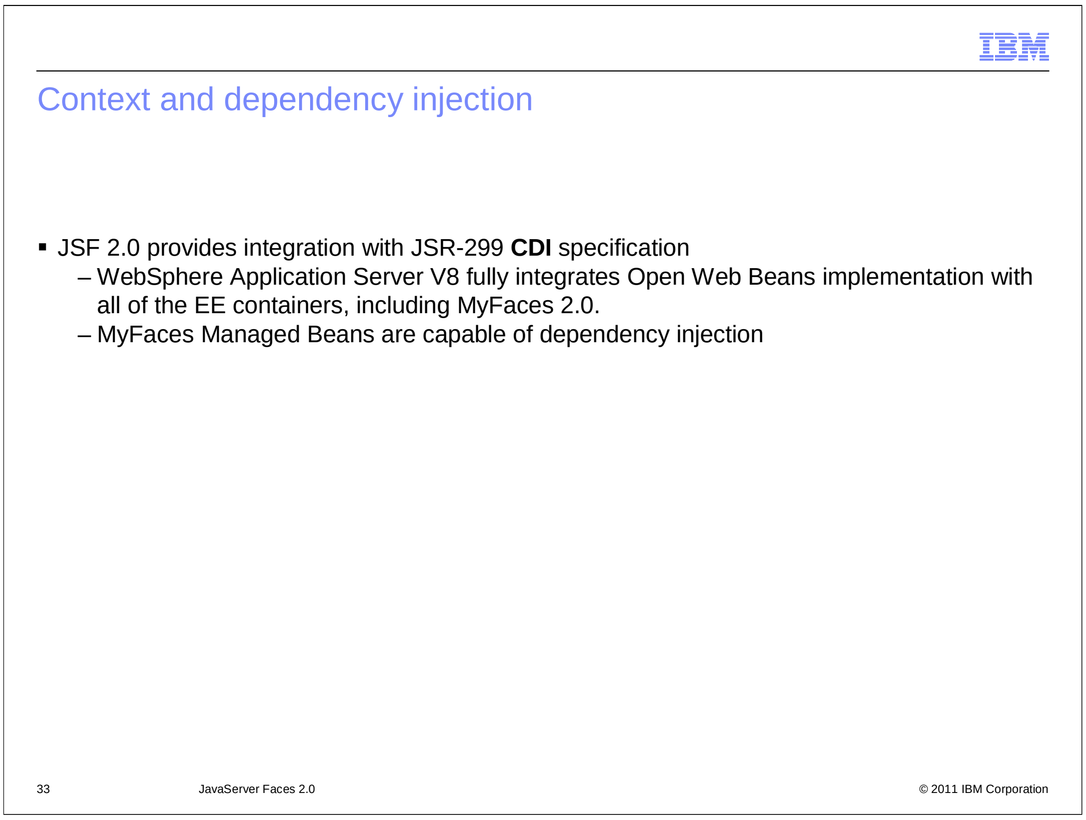
Bean validation

- JSF 2.0 provides integration with JSR-303 **bean validation**, which promises to simplify data validation
- JSR-303 defines a generic mechanism for data validation and includes standard annotations (@Min, @Max, @NotNull, and so on) and custom validation facilities
- The JSF runtime will use JSR-303 bean validation if present, otherwise it will not
- JSF 2.0 also includes support for the <f:validateBean> tag, which is used to specify which bean validation groups are to be used

32 JavaServer Faces 2.0 © 2011 IBM Corporation

JSF 2.0 supports the new JSR-303 Bean Validation specification. All JSF managed beans can be annotated using the new Bean Validation annotations or any custom validators that follow the Bean Validation specification. This allows a developer to create a single set of validators that can be used across any Java EE specification that supports JSR-303 Bean Validation specification, including JSF.

JSF 2.0 also supports a new <f:validateBean> tag to apply the JSR-303 validators to JSF components in a page.



Context and dependency injection

- JSF 2.0 provides integration with JSR-299 **CDI** specification
  - WebSphere Application Server V8 fully integrates Open Web Beans implementation with all of the EE containers, including MyFaces 2.0.
  - MyFaces Managed Beans are capable of dependency injection

33 JavaServer Faces 2.0 © 2011 IBM Corporation

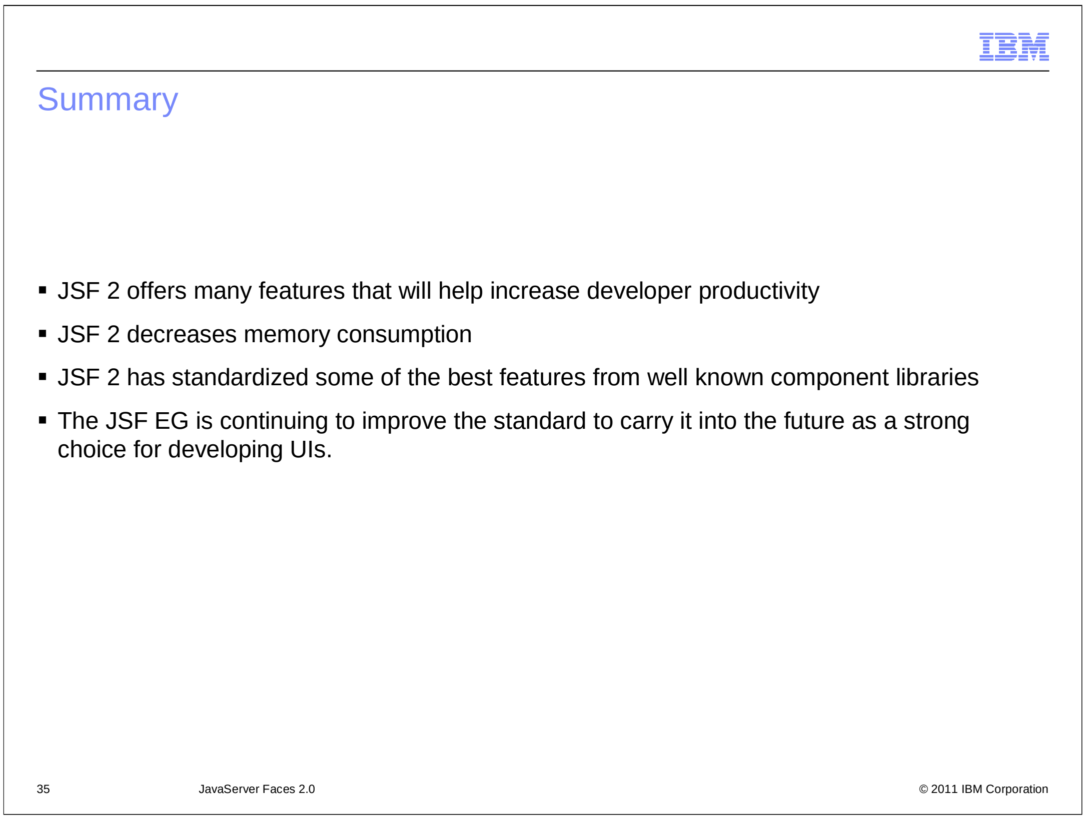
JSF 2.0 allows the use of the new JSR-299 Context and Dependency Injection specification for managed beans. This means a developer can replace the typical JSF managed bean with a CDI managed bean. For WebSphere Application Server V8, Open Web Beans was the CDI implementation used.

Section

Summary

34 JavaServer Faces 2.0 © 2011 IBM Corporation

This section provides a summary of this presentation.

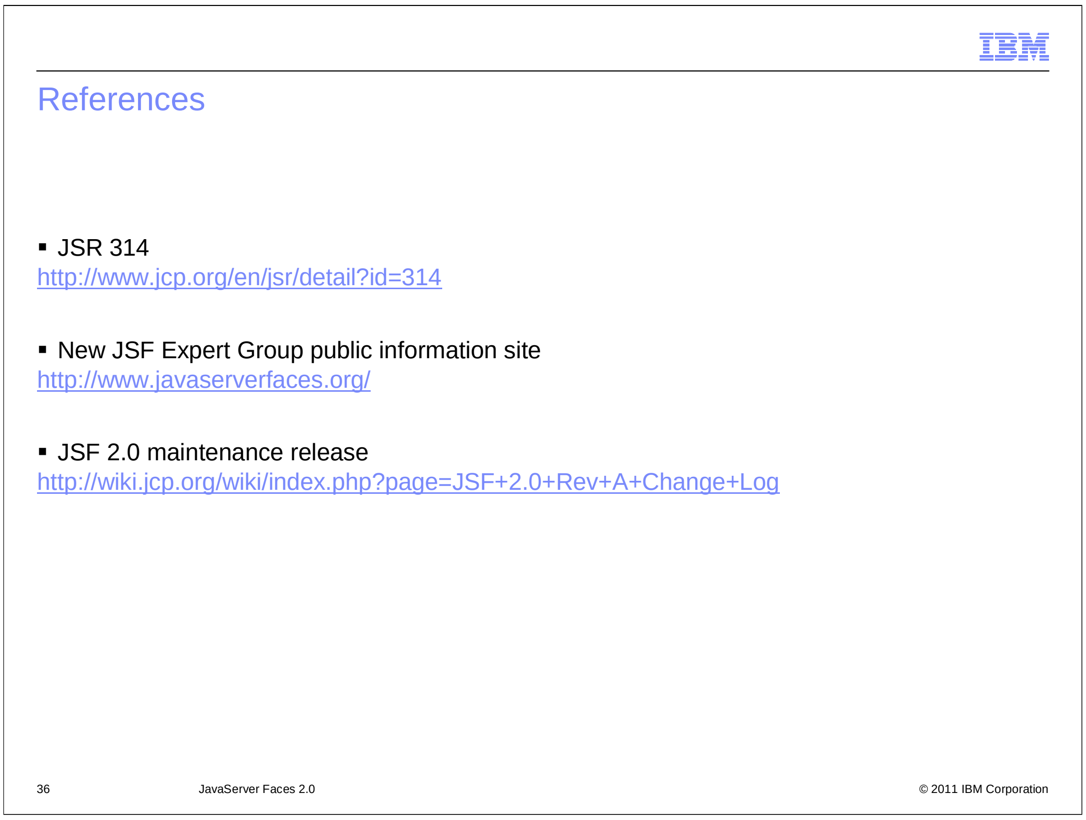


Summary

- JSF 2 offers many features that will help increase developer productivity
- JSF 2 decreases memory consumption
- JSF 2 has standardized some of the best features from well known component libraries
- The JSF EG is continuing to improve the standard to carry it into the future as a strong choice for developing UIs.

35 JavaServer Faces 2.0 © 2011 IBM Corporation

JSF 2.0 brings the best ideas in web application development to the Java EE platform. JSF 2.0 features help maximize the productivity of the web application development experience, minimize the complexity of maintenance of the web application during its production lifetime, make it easy to create responsive rich user interfaces through effective use of Ajax techniques, make it possible to expand the reach of your web application by continuing to support fully functional server based web applications that do not use JavaScript in the client, use modularity to expand integration opportunities between the JSF framework and other client and server side web application technologies and make it easy to validate your data by leveraging Beans Validation.



References

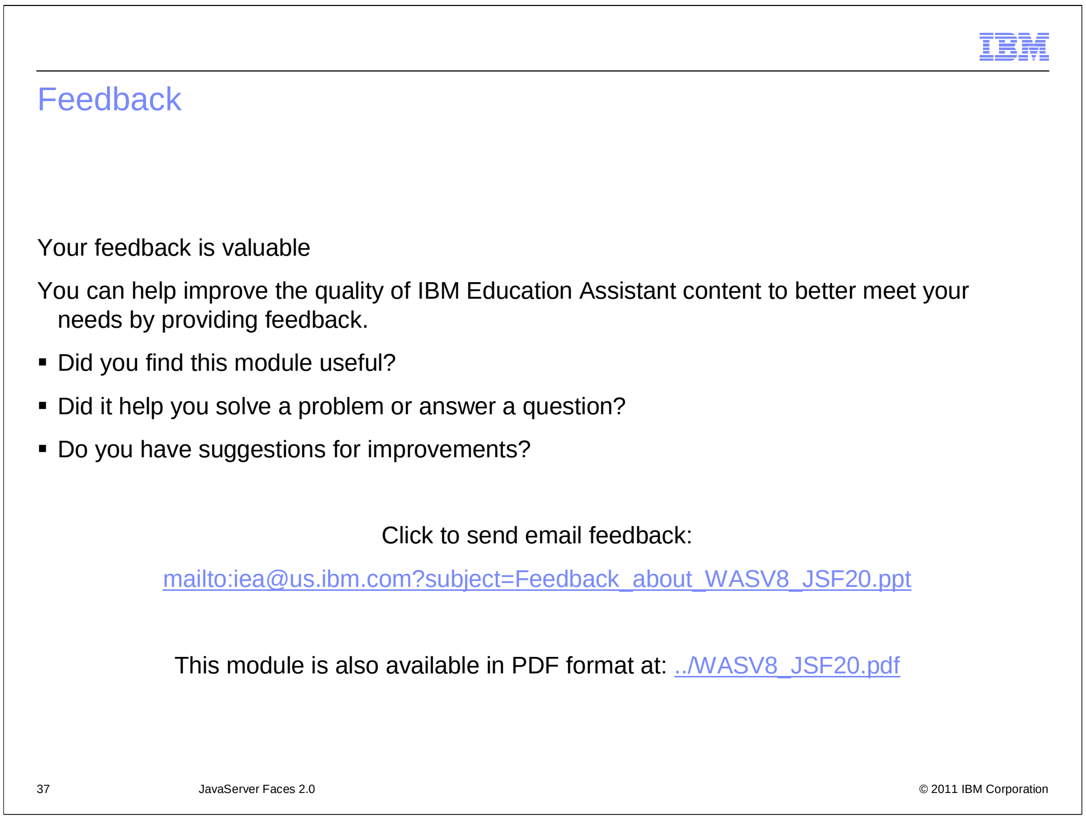
- JSR 314
http://www.jcp.org/en/jsr/detail?id=314

- New JSF Expert Group public information site
http://www.javaserverfaces.org/

- JSF 2.0 maintenance release
http://wiki.jcp.org/wiki/index.php?page=JSF+2.0+Rev+A+Change+Log

36 JavaServer Faces 2.0 © 2011 IBM Corporation

This slide contains links to useful information.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

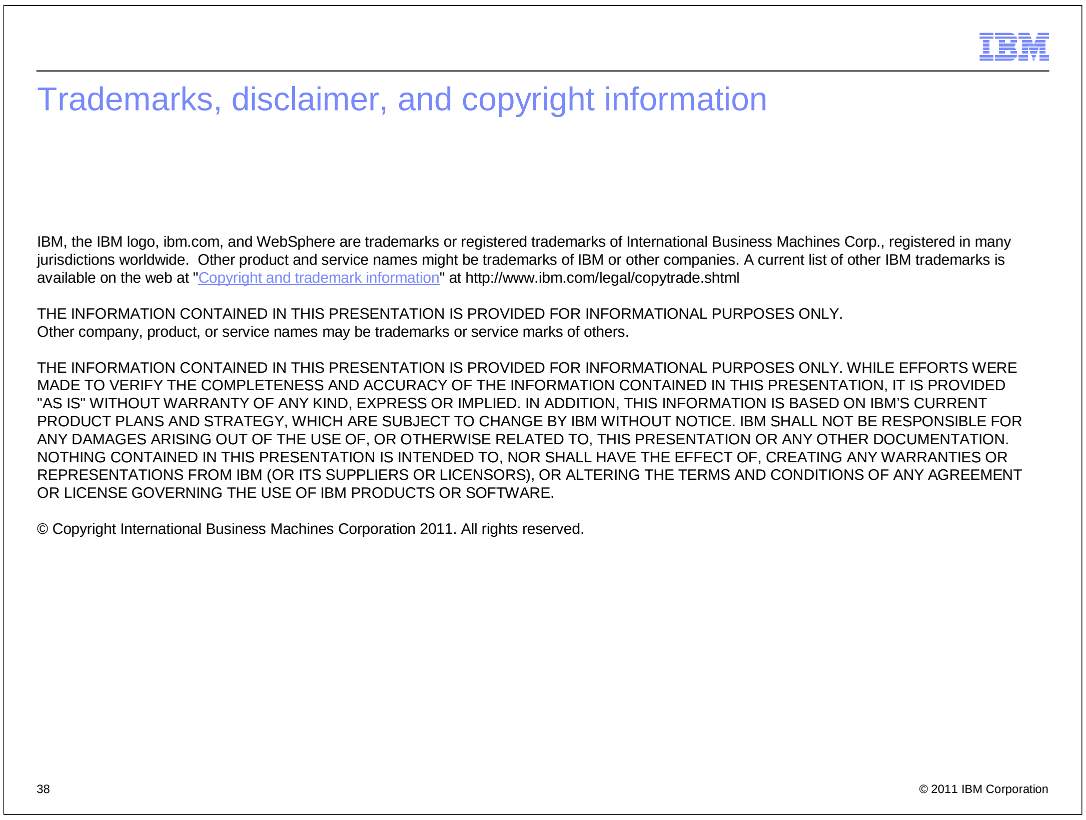
Click to send email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASV8_JSF20.ppt

This module is also available in PDF format at: ../WASV8_JSF20.pdf

37 JavaServer Faces 2.0 © 2011 IBM Corporation

You can help improve the quality of IBM Education Assistant content by providing feedback.



Trademarks, disclaimer, and copyright information