Java Server Faces

# Introduction:

JavaServer Faces technology is a server-side user interface component framework for Java

technology-based web applications.

The main components of JavaServer Faces technology are as follows:

* An API for representing UI components and managing their state; handling events,

server-side validation, and data conversion; defining page navigation; supporting

internationalization and accessibility; and providing extensibility for all these features

■ Two JavaServer Pages (JSP) custom tag libraries for expressing UI components within a JSP

page and for wiring components to server-side objects.

* The well-defined programming model and tag libraries significantly ease the burden of building and maintaining web applications with server-side UIs.
* With minimal effort, you can Drop components onto a page by adding component tags

■ Wire component-generated events to server-side application code

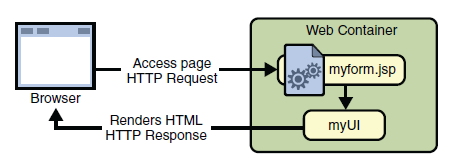
■ Bind UI components on a page to server-side data

■ Construct a UI with reusable and extensible components

■ Save and restore UI state beyond the life of server requests

# JavaServer Faces Technology User Interface

The user interface you create with JavaServer Faces technology runs on the server and renders back to the client.



The JSP page, myform.jsp, is a *JavaServer Faces page*, which is a JSP page that includes

JavaServer Faces tags. It expresses the user interface components by using custom tags defined

by JavaServer Faces technology. The UI for the web application (represented by myUI in the

figure) manages the objects referenced by the JSP page. These objects include

■ The UI component objects that map to the tags on the JSP page

■ Any event listeners, validators, and converters that are registered on the components

■ The JavaBeans components that encapsulate the data and application-specific functionality

of the components

# JavaServer Faces Technology Benefits

One of the greatest advantages of JavaServer Faces technology is that it offers a clean separation

between behavior and presentation.

The separation of logic from presentation also allows each member of a web application

development team to focus on his or her piece of the development process, and it provides a

simple programming model to link the pieces.

Although JavaServer Faces technology includes a JSP custom tag library for

representing components on a JSP page, the JavaServer Faces technology APIs are layered

directly on top of the Servlet API, This layering of APIs enables several

important application use cases, such as using another presentation technology instead of JSP

pages, creating your own custom components directly from the component classes, and

generating output for various client devices.

Most importantly, JavaServer Faces technology provides a rich architecture for managing

component state, processing component data, validating user input, and handling events.

# What Is a JavaServer Faces Application?

For the most part, a JavaServer Faces application is like any other Java web application. A typical

JavaServer Faces application includes the following pieces:

■ A set of JSP pages (although you are not limited to using JSP pages as your presentation

technology)

■ A set of *backing beans*, which are JavaBeans components that define properties and

functions for UI components on a page

■ A application configuration resource file, which defines page navigation rules and

configures beans and other custom objects, such as custom components

■ A deployment descriptor (a web.xml file)

■ Possibly a set of custom objects created by the application developer. These objects might

include custom components, validators, converters, or listeners.

■ A set of custom tags for representing custom objects on the page

A JavaServer Faces application that includes JSP pages also uses the standard tag libraries

defined by JavaServer Faces technology for representing UI components and other objects on

the page.

# Steps in the Development Process

Developing a simple JavaServer Faces application usually requires these tasks:

■ Mapping the FacesServlet instance.

■ Creating the pages using the UI component and core tags.

■ Defining page navigation in the application configuration resource file.

■ Developing the backing beans.

■ Adding managed bean declarations to the application configuration resource file.

## Mapping the FacesServlet Instance

All JavaServer Faces applications must include a mapping to the FacesServlet instance in their

deployment descriptors. The FacesServlet instance accepts incoming requests, passes them to

the life cycle for processing, and initializes resources.

Example:

<servlet>

<display-name>FacesServlet</display-name>

<servlet-name>FacesServlet</servlet-name>

<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>FacesServlet</servlet-name>

<url-pattern>/guess/\*</url-pattern>

</servlet-mapping>

The above file segment represents part of a typical JavaServer Faces web deployment descriptor.

# The Lifecycle of the JSF Application

By default, JavaServer Faces automatically handles most of the lifecycle actions for you. However it also exposes the different stages of the request lifecycle, so that you can modify or perform different actions if your application requirements warrant it.

The lifecycle of a JavaServer Faces application starts and ends with the following activity: The client makes a request for the web page, and the server responds with the page. The lifecycle consists of two main phases: *execute* and *render*.

During the execute phase, several actions can take place: The application view is built or restored, the request parameter values are applied, conversions and validations are performed for component values, backing beans are updated with component values, and application logic is invoked. For a first (initial) request, only the view is built. For subsequent (postback) requests, some or all of the other actions can take place.

In the render phase, the requested view is rendered as a response to the client. Rendering typically, is the process of generating output such as HTML or XHTML that can be read by the client (usually a browser).

The following takes place for a JSF application:

1. When a client makes a first (initial) request for any web page, the application is compiled.
2. The compiled application is executed and a new component tree (UIViewRoot) is constructed for the application and is placed in a FacesContext.
3. The component tree is populated with the component and the backing bean property associated with it.
4. A new view is built based on the component tree.
5. The view is rendered to the requesting client as a response.
6. The component tree is destroyed automatically.
7. On subsequent (postback) requests, the component tree is rebuilt and the saved state is applied.

Introduction to Facelets

The term Facelets is the view declaration language for JavaServer Faces technology.

# What Is Facelets?

Facelets is a powerful but lightweight page declaration language that is used to build JavaServer

Faces views using HTML style templates and to build component trees. Facelets features

include the following:

* Use of XHTML for creating web pages

■ Support for Facelets tag libraries in addition to JavaServer Faces and JSTL tag libraries

■ Support for the Expression Language (EL)

■ Templating for components and pages

Facelets supports these features through the reuse of code and through ease of development,

which are important considerations for developers to adopt JavaServer Faces as the

presentation platform for large-scale projects.

Facelets advantages include the following:

■ Support for code reuse through templating and composite components

■ Functional extensibility of components and other server-side objects through customization

■ Faster compilation time

■ Compile time EL validation

■ High performance rendering

Facelets views are usually created as XHTML pages. JavaServer Faces implementations support

XHTML pages created in conformance with the XHTML TransitionalDocument Type

Definition (DTD), as listed at http://www.w3.org/TR/xhtml1/

#a\_dtd\_XHTML-1.0-Transitional. By convention, web pages built with XHTML have an

.xhtml extension.

JavaServer Faces technology supports various tag libraries to add components to a web page. To

support the JavaServer Faces tag library mechanism, Facelets uses XML namespace

declarations. Table 5–1 lists the tag libraries supported by Facelets.

In addition, Facelets supports tags for composite components for which you can declare custom

prefixes.

Based on the JavaServer Faces support for Expression Language (EL) syntax defined by JSP 2.1,

Facelets uses EL expressions to reference properties and methods of backing beans. EL

expressions can be used to bind component objects or values to methods or properties of

managed beans.

**Developing a Simple Facelets Application**

Developing a simple JavaServer Faces application using Facelets technology, usually requires

these tasks:

■ Developing the backing beans

■ Creating the pages using the component tags

■ Defining page navigation

■ Mapping the FacesServlet instance

■ Adding managed bean declarations

# Creating a Facelets Application

## Developing a Backing Bean

In a typical JavaServer Faces application, each page of the application connects to a backing

bean (a type of managed bean). The backing bean defines the methods and properties that are

associated with the components.

## Creating FaceletsViews

Creating a page or view is the responsibility of a page author. This task involves adding

components on the pages, wiring the components to backing bean values and properties, and

registering converters, validators, or listeners onto the components.

## Configuring the Application

Configuring a JavaServer Faces application involves various tasks which include adding

managed-bean declarations, navigation rules, and resources bundle declarations in the

application configuration resource files such as faces-config.xml, and mapping the Faces

Servlet in the web deployment descriptor file such as a web.xml file.

# Templating

JavaServer Faces 2.0 provides the tools to implement user interfaces that are easy to extend and

reuse. Templating is a useful feature available with Facelets that allows you to create a page that

will act as the base or template for the other pages in an application. By using templates, you can

reuse code and avoid recreating similarly constructed pages. Templating also helps in

maintaining a standard look and feel in an application with a large number of pages.

Table 5–2 lists Facelets tags that are used for templating and their respective functionality.

Tag Function

ui:component Defines a component that is created and added to the component tree.

ui:composition Defines a page composition that optionally uses a template. Content outside of this

tag is ignored.

ui:debug Defines a debug component that is created and added to the component tree.

ui:define Defines content that is inserted into a page by a template.

ui:decorate Similar to the composition tag but does not disregard content outside this tag.

ui:fragment Similar to the component tag but does not disregard content outside this tag.

ui:include Encapsulate and reuse content for multiple pages.

ui:insert Inserts content into a template.

ui:param Used to pass parameters to an included file.

ui:repeat Used as an alternative for loop tags such as c:forEach or h:dataTable.

ui:remove Removes content from a page.

The Facelets tag library includes the main templating tag <ui:insert>. A template page that is

created with this tag, allows defining a default structure for a page. A template page is used as a

template for other pages, usually referred to as a client pages.

The client page invokes the template by using the <ui:composition> tag. In the following

example, a client page named templateclient.xhtml invokes the template page named

template.xhtml, from the preceding example. A client page allows content to be inserted with

the help of the <ui:define> tag.

# Composite Components

JavaServer Faces technology offers the concept of composite components with Facelets. A

composite component can be considered a a special type of template that acts as a component.

Any component essentially is a piece of reusable code that is capable of a certain functionality.

A component also has validators, converters, and listeners attached to it to perform certain defined actions.

A composite component is a component that consists of a collection of markups and other

existing components. It is a reusable, user-created component that is capable of a customized,

defined functionality and can have validators, converters, and listeners attached to it like any

other JavaServer Faces component.

With Facelets, any XHTML page that is inserted with markups and other components can be

converted into a composite component. Using the resources facility, the composite

component can be stored in a library that is available to the application from the defined

resources location.

Expression Language

This chapter introduces the Expression Language (also referred to as EL) which provides an

important mechanism for enabling the presentation layer (web pages) to communicate with the

application logic (backing beans). The EL is used both by JavaServer Faces technology and by

JavaServer Pages (JSP) technology.

The EL represents a union of the expression languages offered by JavaServer Faces technology

and JSP technology.

**Overviewof the EL**

The EL allows page authors to use simple expressions to dynamically access data from

JavaBeans components.

JavaServer Faces technology uses EL for the following functions:

■ Deferred and immediate evaluation of expressions

■ The ability to set as well as get data

■ The ability to invoke methods

To summarize, the EL provides a way to use simple expressions to perform the following tasks:

■ Dynamically read application data stored in JavaBeans components, various data structures,

and implicit objects

■ Dynamically write data, such as user input into forms, to JavaBeans components

■ Invoke arbitrary static and public methods

■ Dynamically perform arithmetic operations

The EL is also used to specify the following kinds of expressions that a custom tag attribute will

accept:

■ *Immediate evaluation expressions* or *deferred evaluation expressions*. An immediate

evaluation expression is evaluated immediately by the underlying technology such as

JavaServer Faces. A deferred evaluation expression can be evaluated later by the underlying

technology using the expression language.

■ *Value expression* or *method expression*. A value expression references data, whereas a

method expression invokes a method.

■ *Rvalue expression* or *lvalue expression*. An rvalue expression can only read a value, whereas

an lvalue expression can both read and write that value to an external object.

**Immediate and Deferred Evaluation Syntax**

The EL supports both immediate and deferred evaluation of expressions. *Immediate evaluation*

means that the expression is evaluated and the result is returned immediately when the page is

first rendered. *Deferred evaluation* means that the technology using the expression language can

employ its own machinery to evaluate the expression sometime later during the page’s lifecycle,

whenever it is appropriate to do so.

Those expressions that are evaluated immediately use the ${} syntax. Expressions whose

evaluation is deferred use the #{} syntax.

Because of its multiphase lifecycle, JavaServer Faces technology mostly uses deferred evaluation

expressions. During the lifecycle, component events are handled, data is validated, and other

tasks are performed in a particular order. Therefore, a JavaServer Faces implementation must

defer evaluation of expressions until the appropriate point in the lifecycle.

Other technologies using the EL might have different reasons for using deferred expressions.

**Immediate Evaluation**

All expressions using the ${} syntax are evaluated immediately. These expressions can only be

used within template text or as the value of a tag attribute that can accept runtime expressions.