What is JSF?

JavaServer Faces (JSF) is a Java framework for building Web applications. It simplifies development by providing a component-centric approach to developing Java Web user interfaces.

JavaServer Faces also appeals to a diverse audience of Java/Web developers. “Corporate developers” and Web designers will find that JSF development can be as simple as dragging and dropping user interface (UI) components onto a page, while “systems developers” will find that the rich and robust JSF API offers them unsurpassed power and programming flexibility.

Java Server Faces also ensures that applications are well designed with greater maintainability by integrating the well established Model-View-Controller (MVC) design pattern into it’s architecture.

What is a component-based?

A component is something that has known attributes and behaviors that is, all the components in JSF have at least a base set of attributes and behaviors in common. This component-based model is beneficial in handling the user actions.

A user action, such as a button click, flows through a defined process from the button click to the business logic that performs the requested process. Hence, it allows you to process client-generated events (For instance, changing the value of a text box or clicking on a button).

JSF has the following

* A set of pre-fabricated UI components.
* An event-driven programming model.
* A component model that enables third-party developers to supply additional components.

It is integrated with MVC design pattern so that applications can be designed well with greater maintainability.

The MVC design pattern splits an application design into three separate parts like

Model: Handles data and logic

It provides a componentized representation of application data (or state) and functional logic. It is essentially the core of the application.

View: Handles output (presentation)

This participant provides one or more representations of application data and behavior in a graphical form for user consumption. This is the only participant that a user directly interacts with.

Controller: Handles processing of an application.

This participant processes user-driven events, which may result in updates to the Model or direct manipulation of the View.

In JSF Framework

* The Java Beans component serves as a Model,
* JSP serves as a View and
* FacesServlet serves as a Controller.

***The WAR file typically has this directory structure:***

index.html

JSP pages

WEB-INF/

-web.xml

-faces-config.xml

-tag library **descriptors** (optional)

-classes/

-**class** **files**

-properties files

-lib/

-JSF JAR files

-JSTL JAR files

-Application JAR files

***JavaServer Faces applications require several JAR files to run properly. These JAR files are as follows:***

* jsf-api.jar (contains the javax.faces.\* API classes)
* jsf-impl.jar (contains the implementation classes of the JavaServer Faces implementation)
* jstl.jar (required to use JSTL tags and referenced by JavaServer Faces implementation classes)
* standard.jar (required to use JSTL tags and referenced by JavaServer Faces reference implementation classes)
* commons-beanutils.jar (utilities for defining and accessing JavaBeans component properties)
* commons-digester.jar (for processing XML documents)
* commons-collections.jar (extensions of the Java 2 SDK Collections Framework)
* commons-logging.jar (a general-purpose, flexible logging facility to allow developers to instrument their code with logging statements)

|  |  |
| --- | --- |
|  | How is JSF different than Struts? |

Struts is an open-source Java Web application framework whose architecture is based on the classical Model View Controller (MVC) design paradigm in which requests are routed through a controller (ActionServlet) that provides overall application management and dispatches the requests to application components.

Struts have become a sort of pseudo standard for designing UI applications because of its usability.

JSF is inspired by the Struts framework and will provide the following components, some of which are similar to components you can find in the Struts framework:

JSF and Struts have many things in common. For example, JSF has FacesServlet for dispatching a request to appropriate components similarly to ActionServlet in Struts.

JSF also provides standard UI components similar to Struts and the tag library components.

JSF is a User-interface framework for Java Web applications where the main focus is on the view tier of an MVC-based architecture, that includes:

* Server-side reusable components for creating user interfaces
* A set of JSP tags for accessing those components
* A framework for storing UI state information beyond the life of the server request
* A framework for developing custom UI components
* The wiring of client-side events to server-side components (FacesServlet)
* The opportunity for IDE vendors to develop standards for a Web application framework
* A standard component API for specifying the state and behaviour of a wide range of components, including simple components, such as input fields, and more complex components such as scrollable data tables
* A separate rendering model that defines how to render the components in various ways. For example, a component used for selecting an item from a list can be rendered as a menu or a set of radio buttons
* An event and listener model that defines how to handle events generated by activating a component, such as when a user clicks a button

JSF encompasses most of what is useful in Struts and goes far beyond it in scope. Here’s a quick list of some differences:

* Naming conventions are different
* JSP: Use Bean tag defines beans IN Web pages. JSF: Managed Beans are defined in an application.
* UI Control-specific behaviour (without added JavaScript!) is supported by JSF (not Struts).
* The UI Control tree to represent a Web page is more refined than the basic Action Form model in Struts.

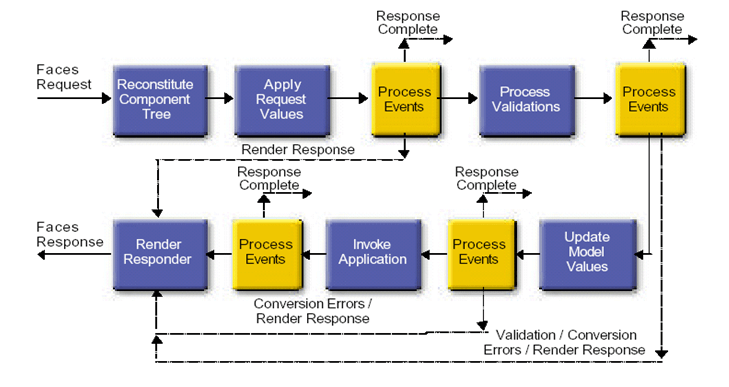
Faces have the following advantages over Struts:

* Eliminated the need for a Form Bean
* Allows the use of the same POJO on all Tiers because of the Backing Bean

A typical JSF application consists of the following parts:

* JavaBeans components for managing application state and behavior.
* Event-driven development (via listeners as in traditional GUI development).
* Pages that represent MVC-style views; pages reference view roots via the JSF component tree.

|  |  |
| --- | --- |
|  | JSF Life cycle |



Life cycle of JSF

The Life cycle of JSF application includes totally six phases

1. Restore view  
2. Apply request values; process events  
3. Process validations; process events  
4. Update model values; process events  
5. Invoke application; process events  
6. Render response

The six phases show the order in which JSF typically processes a form GUI. The list shows the phases in their likely order of execution with event processing at each phase, but the JSF life cycle is hardly set in stone.

Life-cycle handles two types of requests

Initial request & Postback

— Initial request

* A user requests the page for the first time
* Lifecycle only executes the restore view and render response phases

— Postback

* A user submits the form contained on a page that was previously loaded into the browser as a result of executing an initial request
* Lifecycle executes all phases

|  |  |
| --- | --- |
|  | JSF Features |

JSF Features are as below

It provides the following tools to application designers and developers:

* Simple, lightweight classes to represent input events and stateful, server-side GUI components
* A standard GUI component framework for tool integration.
* A set of APIs for: representing UI components and managing their state, handling events and input validation, defining page navigation, and supporting internationalization and accessibility.
* A set of HTML form input elements that represent the server-side GUI components
* A JavaBeans model for translating input events to server-side behaviour
* Validation APIs for both the client side and server side
* Internationalization and localization
* Automatic view presentation, customized to client type (for example, browser or media type)
* Integrated support for accessibility
* A Java Server Pages (JSP) custom tag library for expressing a Java Server Faces interface within a JSP page

|  |  |
| --- | --- |
|  | Advantages of JSF |

Advantages of JSF are as below

The main benefits of Java Server Faces technology include

* Ease-of-Use: Several aspects of the Java Server Faces architecture contribute to its ease-of-use. For one, Java Server Faces technology offers a clean separation  
  between logic and presentation, enabling a wide range of users — from Web-page designers to component developers. It results in a division of labour and a shorter development cycle.
* Also, a user-interface created with JSF handles all the complexities of user-interface management, including input validation, component-state management, page navigation, and event handling.
* It automatically saves the form data and repopulates the form when it is displayed at the client side.
* It encapsulates event handling and component rendering logic from the programmers.
* It provides component-centric, client independent development approach.
* It automatically manages the user interface state between multiple requests and multiple clients.
* Standardization: Java Server Faces technology is being developed through the Java Community Process. Several prominent, respected tool vendors are contributing members to support Java Server Faces technology in their tools.
* Device Independence: Java Server Faces technology is designed to be flexible. By defining only component functionality in extensible UI component classes, the Java Server Faces architecture allows component developers to extend the component classes to generate their own component tag libraries.
* Tree-based Component Model: All the components in JSP page (JSF Tags) are persisted in memory which gives the ability to insert any button or table in the middle of the page at runtime rather than code it in the JSP page.

|  |  |
| --- | --- |
|  | Where JSF fits into three-tier architecture |

Generally, in J2EE Web application, there are at least three logic tiers as below:

* The front-end, or Web, tier
* The application tier
* Database tier

Where JSF fits into three-tier architecture

