

# JSF Architecture

- JSF View Components
- Managed Beans
- Navigation

2 - 1

---



## JSF Requirements

- A Faces application needs:
  - Configuration of the Faces servlet in the Web deployment descriptor
  - A Faces configuration file
  - The JSF JARs in the **WEB-INF/lib** folder or provided by the application server
- The application also typically contains JSPs (or some other view technology) and **managed beans**

2 - 2

---

JEE application servers at version 5 or later include the required JSF libraries so that applications need not configure them in the Web app.

Instead of using JSPs for the view, you can use an alternative technology such as Facelets or Velocity.

## Configuring the Faces Servlet

- In JSF, the Faces servlet acts as a **front controller** and must process **ALL** requests
- You configure the Faces servlet in the standard Web deployment descriptor

web.xml

```
1   <servlet>
2     <servlet-name>Faces Servlet</servlet-name>
3     <servlet-class>
4       javax.faces.webapp.FacesServlet
5     </servlet-class>
6     <load-on-startup>-1</load-on-startup>
7   </servlet>
8   <servlet-mapping>
9     <servlet-name>Faces Servlet</servlet-name>
10    <url-pattern>*.faces</url-pattern>
11  </servlet-mapping>
2 - 3
```

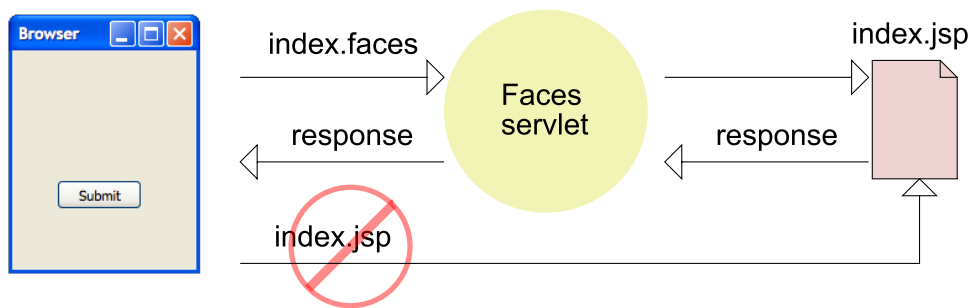
---

You must configure the JSF servlet in web.xml and assign it a URL pattern. There is no requirement for the URL pattern, but the \*.faces extension mapping shown here is common.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://java.sun.com/xml/ns/javaee"
4   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
7   id="WebApp_ID" version="2.5">
8
9   <servlet>
10     <servlet-name>Faces Servlet</servlet-name>
11     <servlet-class>
12       javax.faces.webapp.FacesServlet
13     </servlet-class>
14     <load-on-startup>-1</load-on-startup>
15   </servlet>
16   <servlet-mapping>
17     <servlet-name>Faces Servlet</servlet-name>
18     <url-pattern>*.faces</url-pattern>
19   </servlet-mapping>
20 </web-app>
```

## JSF URLs

- URLs from the client must follow the mapping that you configure in the Web deployment descriptor for the Faces servlet
- In this case, the Faces servlet strips the ".faces" from the URL and substitutes ".jsp" to find a JSP with that name



2 - 4

---

For JSF to work properly, all requests from the client must be routed through the Faces servlet. The servlet will use the input URL to find the JSP for the request.

In this case, given the servlet configuration on the last page, the Faces servlet will process "index.jsp" when the browser submits "index.faces".

There's nothing special about "\*.faces" either -- you can use a different extension or so-called "path mapping" instead.

Also note that it's important that clients do NOT directly access a JSF JSP, since it won't work properly if they do.

## Optional: Preventing Direct JSP Access

- Faces JSPs must be processed by the Faces servlet and should not be accessed directly from a browser
- To disallow direct access, you can write a **security-constraint** element in the Web deployment descriptor

### web.xml

```
1  <security-constraint>
2    <web-resource-collection>
3      <web-resource-name>Hide JSPs</web-resource-name>
4      <url-pattern>/index.jsp</url-pattern>
5      <url-pattern>/addstudent.jsp</url-pattern>
6      . . .
7    </web-resource-collection>
8    <auth-constraint>
9      <description>
10         Since we define no roles, no direct access
11         from the client
12      </description>
13    </auth-constraint>
14 </security-constraint>
```

2 – 5

---

This step is optional, but it's a good idea since it prevents clients from directly accessing JSPs and bypassing the Faces servlet. Note for this to really work, you need to write an entry for EVERY JSP in the Web application.

Also note that if the client does try to access a JSP directly with this in place, they client may get a security exception response that they might not understand.

## Optional: Providing a Start Page

- JEE Web applications can define a **welcome-file** that acts as the start page of the application
- In JSF, the welcome JSP typically forwards to the first "real" page in the application, via the Faces servlet

web.xml

```
<welcome-file-list>  
  <welcome-file>start.jsp</welcome-file>  
</welcome-file-list>
```

start.jsp

```
<jsp:forward page="/index.faces"/>
```

2 – 6

---

With the JEE Web application "welcome-file" mechanism, the application server directly invokes the welcome file, bypassing the Faces servlet, which doesn't work properly.

To have a welcome file that works properly, define the welcome file in the Web deployment descriptor and have the JSP simply forward to the "real" start page, specifying a URL that's processed by the Faces servlet.



## JSF Tag Libraries

- JSF defines two tag libraries you can use in JSPs:
- The **core** library is view-independent and lets you perform validation, data conversion and so forth
- The **HTML** library lets you define input forms – you should use this library instead of standard HTML tags

```
<%@taglib  
    uri="http://java.sun.com/jsf/core" prefix="f"%>
```

```
<%@taglib  
    uri="http://java.sun.com/jsf/html" prefix="h"%>
```

## Sample JSF Page

```
1  <%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
2  <%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
3  <html>
4      <head>
5          <title>Add Student</title>
6      </head>
7      <f:view>
8          <body>
9              <h:form>
10                 <p>Student name:<h:inputText
11                     value="#{student.name}"/></p>
12                 <p>Student ID: <h:inputText
13                     value="#{student.id}"/></p>
14                 <p>GPA: <h:inputText value="#{student.gpa}"/></p>
15                 <p><h:commandButton value="Submit"
16                     action="#{student.addStudent}"/></p>
17             </h:form>
18         </body>
19     </f:view>
20 </html>
```

2 - 8

---

## Managed Beans

- JSF applications use **managed beans** to represent business data and logic
- Managed beans expose **properties** via get/set methods
- You configure managed beans in the Faces configuration file and specify their **scope**

1	public class Student	
2	{	faces-config.xml
3	. . .	<managed-bean>
4	public int getId()	<managed-bean-name>
5	{	student
6	. . .	</managed-bean-name>
7	}	<managed-bean-class>
8	public void setId(int i)	univ.Student
9	{	</managed-bean-class>
10	. . .	<managed-bean-scope>
11	}	request
12	}	</managed-bean-scope>
2 - 9		</managed-bean>

---

Managed beans expose properties by defining get/set methods and can provide logic by providing methods.

A JSF page can access a managed bean's properties and methods using the JSF Expression Language.

# Introduction to the JSF Expression Language

- JSF provides an **expression language** so that page authors can access managed bean properties as well as several predefined objects (e.g. **cookie**)
- JSF version 1.2 supports the **Unified Expression Language**, which unifies the JSF EL with the standard JSP expression language
- JSF expressions reside within JSF tags and have basic syntax **`#{expr}`**

`#{student.id}`

`#{student.gpa}`

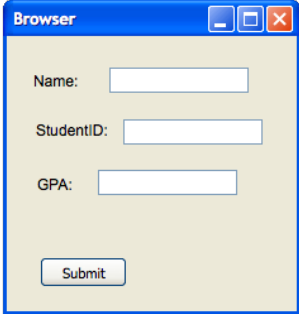
2 – 10

---

When you write a JSF expression that references a property, behind the scenes, JSF calls the bean's get/set methods.

# Introduction to JSF Components

- A JSF page comprises one or more **components**, each of which displays content and/or accepts user input
- The Faces servlet uses JSF tags in the JSP to build a **component tree** of objects on the server
- The standard JSF components generate HTML

<u>JSF Tag</u>		<u>Component class</u>
<h:form>		UIForm
<h:inputText>		UIInput
<h:inputText>		UIInput
<h:inputText>		UIInput
<h:commandButton>		UICommand

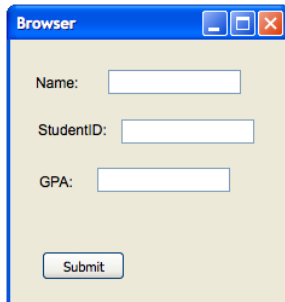
2 - 11

---

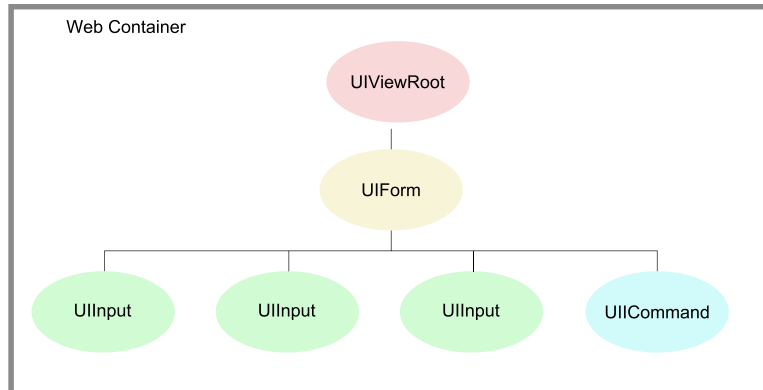
In a JSF JSP, you use JSF custom actions (tags) to define the form. At runtime, the JSF controller builds a component tree from the custom actions, using component classes defined by JSF.

## The Component Tree

- When the Faces servlet processes a request, it first builds a **component tree** and lets the tree participate in processing the request and generating a response



A screenshot of a web browser window titled "Browser". The window contains a form with three input fields: "Name:", "StudentID:", and "GPA:". Below the input fields is a "Submit" button.

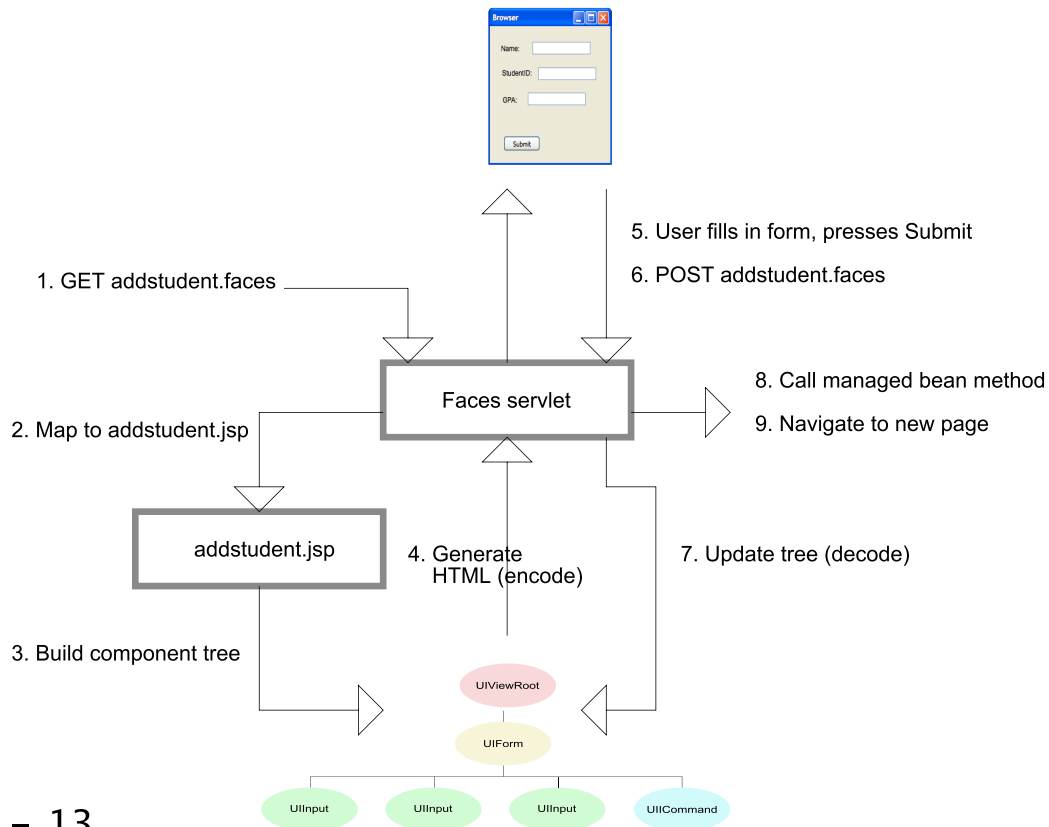


2 - 12

---

Components are the heart of a JSF application -- they provide the user interface. JSF borrowed this notion from client-side GUIs like Visual Basic or Java Swing.

# Typical Request Processing



2 - 13

To start this flow, the user first enters the URL for a page, which typically has an extension of .faces. This invokes the Faces servlet, which finds the matching JSP and invokes the JSP.

The JSF actions in the JSP collaborate to build the component tree. The Faces servlet then renders the JSP and component tree. All non-JSF text (e.g. HTML paragraphs) are passed back verbatim, and the JSF components render their HTML equivalent. For example, the `h:inputText` tag's component generates an HTML tag like `<input type="text">`. This process is known as encoding.

The user then fills in the form and submits it back to the same URL. The Faces servlet finds the component tree and lets the tree work with the user input, a process known as decoding. The result is that each component captures its tag's user input and stores it in the specified property in the managed bean.

The Faces servlet then invokes the method specified by the Submit button's JSF expression. The method processes the request (examining the user-input) and returns a string which the servlet uses to navigate to a new page.

## Invoking an Application Method

- When the user submits the form, the Faces servlet lets the component tree **decode** it and update managed-bean properties, then invokes the method specified on the submit button

```
<h:commandButton
  value="Add student"
  action="#{student.addStudent}"/>

    private int id;
    private double gpa;
    private String name;
    . . .

    public String addStudent
    {
        // process id,gpa,name as required
        // store objects at request or session scope

        if (ok) return "success"
        else return "bad-news";
    }
```

2 - 14

---

Before calling "addStudent", the framework has initialized the "id", "gpa" and "name" properties in the managed bean, so "addStudent" can use those values to perform data processing.

Based on the results of processing, the "addStudent" method returns a mapping string -- the Faces servlet uses this string to determine to which page to navigate.



## Accessing the Servlet Environment

- The processing method in the managed bean typically accepts no parameters
- If the method needs to access the JSF and/or servlet environment (e.g. the servlet request), it can retrieve the JSF **external context**

```
1 ExternalContext context =
2   FacesContext.getCurrentInstance().getExternalContext();
3
4 HttpServletRequest request =
5   (HttpServletRequest)context.getRequest();
6
7 Cookie[] cookies = request.getCookies();
```

2 – 15

---

Often when processing a request, the method needs to work with the servlet or JSF environment. To gain access, the method can retrieve an external context and use it to retrieve objects.

For example, here we show using the ServletRequest object to retrieve the array of HTTP cookies passed on a request.

## Introduction to Navigation

- You can configure **navigation rules** in the Faces configuration file
- The processing method returns a logical string that Faces uses to navigate to another page

```
1    <navigation-rule>
2        <from-view-id>/addstudent.jsp</from-view-id>
3        <navigation-case>
4            <from-outcome>success</from-outcome>
5            <to-view-id>/confirmation.jsp</to-view-id>
6        </navigation-case>
7        <navigation-case>
8            <from-outcome>bad-news</from-outcome>
9            <to-view-id>/failure.jsp</to-view-id>
10       </navigation-case>
11    </navigation-rule>
```

2 - 16

---

## Chapter Summary

In this chapter, you learned:

- How to configure JSF
- How JSF uses URLs
- The fundamental JSF processing model

