JavaServer Faces Technology

JavaServer Faces technology is a server-side component framework for building Java technology-based web applications.

JavaServer Faces technology consists of the following:

- An API for representing 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
- Tag libraries for adding components to web pages and for connecting components to server-side objects

JavaServer Faces technology provides a well-defined programming model and various tag libraries.

These features significantly ease the burden of building and maintaining web applications with server-side user interfaces (UIs).

With minimal effort, you can complete the following tasks.

- Create a web page.
- Drop components onto a web page by adding component tags.
- Bind components on a page to server-side data.
- Wire component-generated events to server-side application code.
- Save and restore application state beyond the life of server requests.
- Reuse and extend components through customization.

This chapter provides an overview of JavaServer Faces technology.

After explaining what a JavaServer Faces application is and reviewing some of the primary benefits of using JavaServer Faces technology, this chapter describes the process of creating a simple JavaServer Faces application.

This chapter also introduces the JavaServer Faces lifecycle by describing the example JavaServer Faces application progressing through the lifecycle stages.

The following topics are addressed here:

What Is a JavaServer Faces Application?

The functionality provided by a JavaServer Faces application is similar to that of any other Java web application.

A typical JavaServer Faces application includes the following parts:

- A set of web pages in which components are laid out
- A set of tags to add components to the web page
- A set of backing beans, which are JavaBeans components that define properties and functions for components on a page

- A web deployment descriptor (`web.xml` file)
- Optionally, one or more application configuration resource files, such as a `faces-config.xml` file, which can be used to define page navigation rules and configure beans and other custom objects, such as custom components

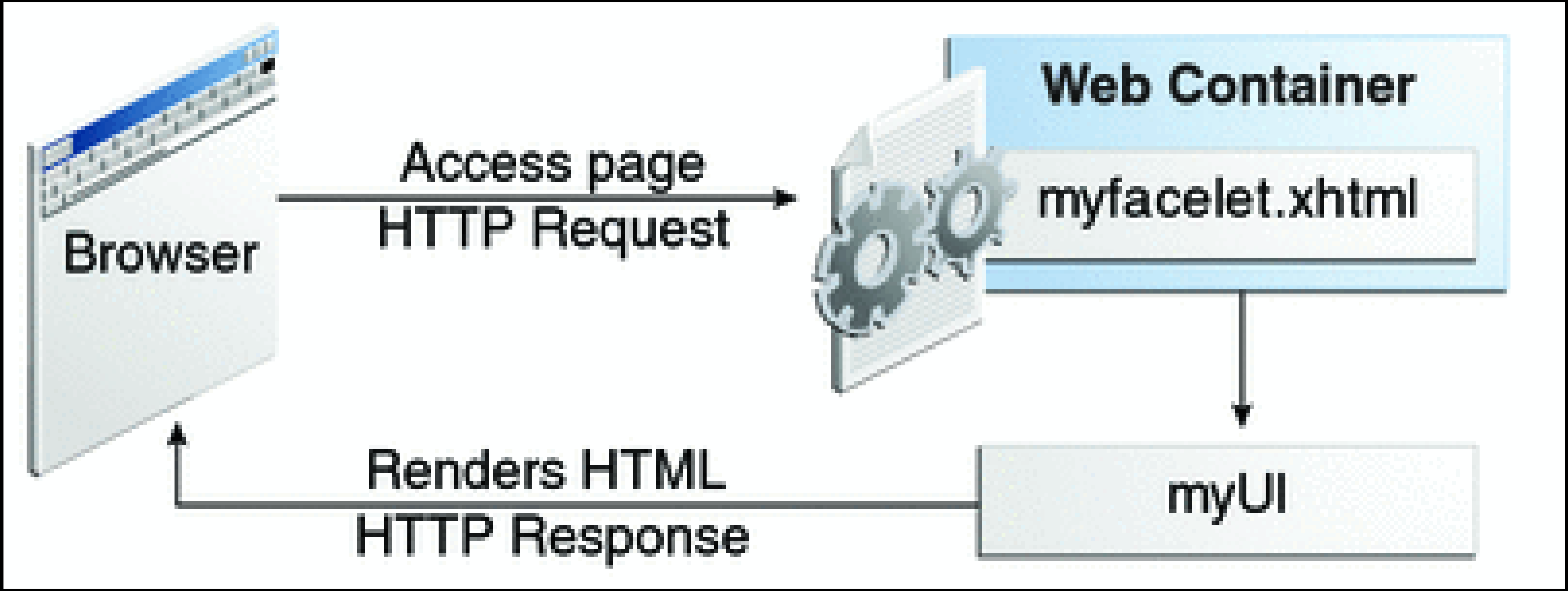
- Optionally, a set of custom objects, which can include custom components, validators, converters, or listeners, created by the application developer

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

Figure 4-1 shows the interaction between client and server in a typical JavaServer Faces application.

In response to a client request, a web page is rendered by the web container that implements JavaServer Faces technology.

Figure 4-1 Responding to a Client Request for a JavaServer Faces Page



The web page, `myfacelet.xhtml`, is built using JavaServer Faces component tags.

Component tags are used to add components to the `view` (represented by `myUI` in the diagram), which is the server-side representation of the page.

In addition to components, the web page can also reference objects, such as the following:

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

- The JavaBeans components that capture the data and process the application-specific functionality of the components

On request from the client, the view is rendered as a response.

Rendering is the process whereby, based on the server-side view, the web container generates output, such as HTML or XHTML, that can be read by the client, such as a browser.

JavaServer Faces Technology Benefits

One of the greatest advantages of JavaServer Faces technology is that it offers a clean separation between behavior and presentation for web applications.

A JavaServer Faces application can map HTTP requests to component-specific event handling and manage components as stateful objects on the server.

JavaServer Faces technology allows you to build web applications that implement the finer-grained separation of behavior and presentation that is traditionally offered by client-side UI architectures.

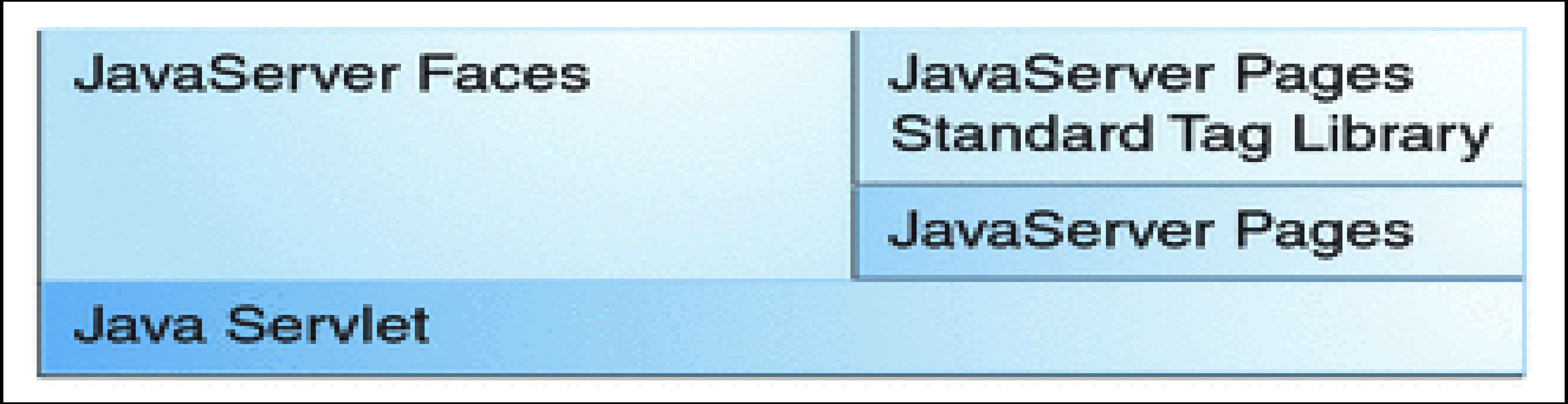
The separation of logic from presentation also allows each member of a web application development team to focus on a single piece of the development process and provides a simple programming model to link the pieces.

For example, page authors with no programming expertise can use JavaServer Faces technology tags in a web page to link to server-side objects without writing any scripts.

Another important goal of JavaServer Faces technology is to leverage familiar component and web-tier concepts without limiting you to a particular scripting technology or markup language.

JavaServer Faces technology APIs are layered directly on top of the Servlet API, as shown in Figure 4-2.

Figure 4-2 Java Web Application Technologies



This layering of APIs enables several important application use cases, such as using different presentation technologies, creating your own custom components directly from the component classes, and generating output for various client devices.

Facelets technology, available as part of JavaServer Faces 2.0, is now the preferred presentation technology for building JavaServer Faces technology-based web applications.

For more information on Facelets technology features, see Chapter 5, Introduction to Facelets.

Facelets technology offers several advantages.

- Code can be reused and extended for components through the templating and composite component features.

- When you use the JavaServer Faces Annotations feature, you can automatically register the backing bean as a resource available for JavaServer Faces applications.

In addition, implicit navigation rules allow developers to quickly configure page navigation.

These features reduce the manual configuration process for applications.

- Most important, JavaServer Faces technology provides a rich architecture for managing component state, processing component data, validating user input, and handling events.

Creating a Simple JavaServer Faces Application

JavaServer Faces technology provides an easy and user-friendly process for creating web applications.

Developing a simple JavaServer Faces application typically requires the following tasks:

- Developing backing beans
- Adding managed bean declarations
- Creating web pages using component tags
- Mapping the `FacesServlet` instance

This section describes those tasks through the process of creating a simple JavaServer Faces Facelets application.

The example is a Hello application that includes a backing bean and a web page.

When accessed by a client, the web page prints out a `Hello World` message.

The example application is located in the directory *tut-install*`/examples/web/hello`.

The tasks involved in developing this application can be examined by looking at the application components in detail.

Developing the Backing Bean

As mentioned earlier in this chapter, a backing bean, a type of managed bean, is a JavaBeans component that is managed by JavaServer Faces technology.

Components in a page are associated with backing beans that provide application logic.

The example backing bean, `Hello.java`, contains the following code:

```
package hello;
import
javax.faces.bean.ManagedBean;
@ManagedBean
public class Hello {
final String world =
"Hello World!";
public String getworld()
{ return world; }
}
```

The example backing bean sets the value of the variable `world` with the string `"Hello World!"`.

The `@ManagedBean` annotation registers the backing bean as a resource with the JavaServer Faces implementation.

For more information on managed beans and annotations, see Chapter 9, Developing with JavaServer Faces Technology.

Creating the Web Page

In a typical Facelets application, web pages are created in XHTML.

The example web page, `beanhello.xhtml`, is a simple XHTML page.

It has the following content:

```
<html
xmlns=
"http://www.w3.org/1999/xhtml"
xmlns:h=
"http://java.sun.com/jsf/html"
>
<h:head>
<title>Facelets Hello World</title>
</h:head>
<h:body>#{hello.world}</h:body>
</html>
```

A Facelets XHTML web page can also contain several other elements, which are covered later in this tutorial.

The web page connects to the backing bean through the Expression Language (EL) value expression `#{hello.world}`, which retrieves the value of the `world` property from the backing bean `Hello`.

Note the use of `hello` to reference the backing bean `Hello`.

If no name is specified in the `@ManagedBean` annotation, the backing bean is always accessed with the first letter of the class name in lowercase.

For more information on using EL expressions, see Chapter 6, Expression Language.

For more information about Facelets technology, see Chapter 5, Introduction to Facelets.

For more information about the JavaServer Faces programming model and building web pages using JavaServer Faces technology, see Chapter 7, Using JavaServer Faces Technology in Web Pages.

Mapping the `FacesServlet` Instance

The final task requires mapping the `FacesServlet`, which is done through the web deployment descriptor (`web.xml`).

A typical mapping of `FacesServlet` is as follows:

```
<servlet>
<servlet-name>Faces
Servlet</servlet-name>
<servlet-class>
javax.faces.webapp.FacesServlet
</servlet-class>
<load-on-startup>
1
</load-on-startup>
</servlet>
```

```
<servlet-mapping>
<servlet-name>
Faces Servlet
</servlet-name>
<url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

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

The web deployment descriptor can also contain other content relevant to a JavaServer Faces application configuration, but that information is not covered here.

Mapping the `FacesServlet` is automatically done for you if you are using an IDE such as NetBeans IDE.

The Lifecycle of the `hello` Application

Every web application has a lifecycle.

Common tasks, such as handling incoming requests, decoding parameters, modifying and saving state, and rendering web pages to the browser, are all performed during a web application lifecycle.

Some web application frameworks hide the details of the lifecycle from you, whereas others require you to manage them manually.

By default, JavaServer Faces automatically handles most of the lifecycle actions for you.

However, it also exposes the various stages of the request lifecycle, so that you can modify or perform different actions if your application requirements warrant it.

It is not necessary for the beginning user to understand the lifecycle of a JavaServer Faces application, but the information can be useful for creating more complex applications.

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.
- 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 is typically the process of generating output, such as HTML or XHTML, that can be read by the client, usually a browser.

The following short description of the example JavaServer Faces application passing through its lifecycle summarizes the activity that takes place behind the scenes.

The `hello` example application goes through the following stages when it is deployed on the GlassFish Server.

1. When the `hello` application is built and deployed on the GlassFish Server, the application is in an uninitiated state.

2. When a client makes an initial request for the `beanhello.xhtml` web page, the `hello` Facelets application is compiled.

3. The compiled Facelets application is executed, and a new component tree is constructed for the `hello` application and is placed in a `FacesContext`.

4. The component tree is populated with the component and the backing bean property associated with it, represented by the EL expression `hello.world`.

5. A new view is built, based on the component tree.

6. The view is rendered to the requesting client as a response.

7. The component tree is destroyed automatically.

8. On subsequent (postback) requests, the component tree is rebuilt, and the saved state is applied.

For more detailed information on the JavaServer Faces lifecycle, see the JavaServer Faces Specification, Version 2.0.

To Build, Package, Deploy, and Run the Application in NetBeans IDE

1. From the File menu, choose Open Project.

2. In the Open Project dialog box, navigate to:
   `tut-install``/examples/web`

3. Select the `hello` folder.

4. Select the Open as Main Project check box.

5. Click Open Project.

6. In the Projects tab, right-click the `hello` project and select Run.

This step compiles, assembles, and deploys the application and then brings up a web browser window displaying the following URL:

```
http://localhost:8080/hello
```

The output looks like this:

```
Hello World!
```

Further Information about JavaServer Faces Technology

For more information on JavaServer Faces technology, see

- JavaServer Faces 2.0 specification:

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

- JavaServer Faces technology web site:

  http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

- JavaServer Faces 2.0 technology download web site:

  http://www.oracle.com/technetwork/java/javaee/download-139288.html

- Mojarra (JavaServer Faces 2.0) Release Notes:

  http://javaserverfaces.java.net/nonav/rlnotes/2.0.0/index.htm