Herramientas de trabajo (3)

Miguel Angel Piña Avelino

Ingeniería de Software,
Facultad de Ciencias, UNAM

27 de febrero de 2018

Índice

Usando Git

Agregando repositorios remotos

Hay que crear un repositorio remoto (en *github*) y agregamos la siguiente instrucción:

```
$ git remote add origin https://github.com/miguelpinia/something.git
```

Hay que remplazar la url por la del repositorio. Una vez agregado el repositorio remoto, hay que empujar todos los cambios que hemos hecho en nuestro repositorio local:

```
git push -u origin master
```

Obteniendo cambios desde repositorio

Supongamos que alguien hizo cambios en el repositorio y los empujó al repositorio, para obtener esos cambios, ejecutamos la instrucción:

```
$ git pull
```

Para ver cuales fueron los cambios que se hicieron entre el último commit que teníamos y el que obtuvimos ejecutamos:

```
$ git diff HEAD
```

Creando ramas

Para agregar cambios sin modificar el contenido de la rama principal (rama master), vamos a crear una nueva rama donde podamos trabajar. Para hacer esto, realizamos lo siguiente:

```
$ git checkout -b mi_nueva_rama
```

A partir de aquí podemos realizar nuevos cambios. Para regresar o moverse entre ramas, usando el comando checkout sin banderas.

```
$ git checkout master
```

Mezclando ramas

Después de realizar una serie de cambios, es necesario que se mezclen con la rama principal. Una forma limpia de hacerlo, es haciendo un rebase de la rama con la que estamos trabajando respecto a la que vamos a mezclar.

Mezclando ramas

La acción de rebase, se refiere a desplazar todos los cambios que existan en la rama a mezclar y que no se han integrado con la rama que estamos trabajando.

```
$ git checkout mi_nueva_rama
$ git rebase master
$ git checkout master
$ git merge --no-ff mi_nueva_rama
$ git push
```

Eliminando ramas

La forma sencilla de eliminar ramas dentro de git es la siguiente:

```
$ git branch -d mi_nueva_rama
```

Java Server Faces

Java Server Faces

JSF es un framework MVC (Modelo-Vista-Controlador) basado en el API de Servlets que proporciona un conjunto de componentes en forma de etiquetas definidas en páginas XHTML mediante el framework Facelets. Antes de la especificación actual se utilizaba JSP para componer las páginas JSF.

Java Server Faces

Entrando un poco más en detalle, JSF proporciona las siguientes características destacables:

- Definición de las interfaces de usuario mediante vistas que agrupan componentes gráficos.

Java Server Faces

Entrando un poco más en detalle, JSF proporciona las siguientes características destacables:

- Definición de las interfaces de usuario mediante vistas que agrupan componentes gráficos.
- Conexión de los componentes gráficos con los datos de la aplicación mediante los denominados beans gestionados.

Java Server Faces

Entrando un poco más en detalle, JSF proporciona las siguientes características destacables:

- Definición de las interfaces de usuario mediante vistas que agrupan componentes gráficos.
- Conexión de los componentes gráficos con los datos de la aplicación mediante los denominados beans gestionados.
- Conversión de datos y validación automática de la entrada del usuario.

Java Server Faces

Entrando un poco más en detalle, JSF proporciona las siguientes características destacables:

- Definición de las interfaces de usuario mediante vistas que agrupan componentes gráficos.
- Conexión de los componentes gráficos con los datos de la aplicación mediante los denominados beans gestionados.
- Conversión de datos y validación automática de la entrada del usuario.
- Navegación entre vistas.

Java Server Faces

Entrando un poco más en detalle, JSF proporciona las siguientes características destacables:

- Definición de las interfaces de usuario mediante vistas que agrupan componentes gráficos.
- Conexión de los componentes gráficos con los datos de la aplicación mediante los denominados beans gestionados.
- Conversión de datos y validación automática de la entrada del usuario.
- Navegación entre vistas.
- Internacionalización

Java Server Faces

Entrando un poco más en detalle, JSF proporciona las siguientes características destacables:

- Definición de las interfaces de usuario mediante vistas que agrupan componentes gráficos.
- Conexión de los componentes gráficos con los datos de la aplicación mediante los denominados beans gestionados.
- Conversión de datos y validación automática de la entrada del usuario.
- Navegación entre vistas.
- Internacionalización
- A partir de la especificación 2.0 un modelo estándar de comunicación Ajax entre la vista y el servidor.

Otras características

Otras características que tiene JSF son:

- Soporte para Ajax

Otras características

Otras características que tiene JSF son:

- Soporte para Ajax
- Componentes múltiples

Otras características

Otras características que tiene JSF son:

- Soporte para Ajax
- Componentes múltiples
- Integración con Facelets

Otras características

Otras características que tiene JSF son:

- Soporte para Ajax
- Componentes múltiples
- Integración con Facelets
- Gestión de recursos (hojas de estilo, imágenes, etc.)

Otras características

Otras características que tiene JSF son:

- Soporte para Ajax
- Componentes múltiples
- Integración con Facelets
- Gestión de recursos (hojas de estilo, imágenes, etc.)
- Facilidad de desarrollo y despliegue

Java Server Faces

Como se ejecuta

Tal y como hemos comentado, JSF se ejecuta sobre la tecnología de Servlets y no requiere ningún servicio adicional, por lo que para ejecutar aplicaciones JSF sólo necesitamos un contenedor de servlets tipo Tomcat o Jetty.

Java Server Faces

Implementaciones

JSF es una especificación y, como tal, existen distintas implementaciones. Sun siempre proporciona una implementación de referencia de las tecnologías Java, que incluye en el servidor de aplicaciones GlassFish. En el caso de JSF, la implementación de referencia s las dos implementaciones más usadas son:

- Mojarra
- MyFaces

Primefaces

PrimeFaces es una librería de componentes para JavaServer Faces (JSF) de código abierto que cuenta con un conjunto de componentes enriquecidos que facilitan la creación de las aplicaciones web. Primefaces está bajo la licencia de Apache License V2. Una de las ventajas de utilizar Primefaces, es que permite la integración con otros componentes como por ejemplo RichFaces.

Documentación

JavaServer Faces Technology

https://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm

Primefaces

http://primefaces.org/

Integrando JSF

Integrando JSF en Maven

Para ejemplificar la integración de JSF y la facilidad con la que se puede operar con JSF, vamos a implementar un formulario que permita simular el registro de un usuario.

Integrando JSF en Maven

Lo primero que hay que hacer es tomar el ejemplo de Maven que se construyó en sesiones pasadas y modificar el archivo `pom.xml` en la sección de dependencias agregando el siguiente código para integrar las dependencias de JSF a nuestro proyecto:

```
<!-- Agrega mos soporte para java server faces -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>
<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-api</artifactId>
  <version>2.2.8</version>
  <type>jar</type>
</dependency>
```

Integrando JSF en Maven

```
<dependency>
  <groupId>com.sun.faces</groupId>
  <artifactId>jsf-impl</artifactId>
  <version>2.2.8-19</version>
</dependency>
<!-- Soporte para primefaces -->
<dependency>
  <groupId>org.primefaces</groupId>
  <artifactId>primefaces</artifactId>
  <version>6.0</version>
</dependency>
```

Integrando JSF con Maven

El siguiente archivo a modificar es web.xml de modo que luzca como el siguiente código

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1"
         xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
  <display-name>Archetype Created Web Application</display-name>

  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

Integrando JSF con Maven

```
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
    <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>servlet</servlet-name>
    <servlet-class>com.miguel.proyecto.MiServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>servlet</servlet-name>
    <url-pattern>/foo</url-pattern>
  </servlet-mapping>
</web-app>
```

Integrando JSF con Maven

Ahora que ya tenemos el soporte de las bibliotecas de JSF dentro del proyecto, vamos a comenzar a crear la página para validar la creación de inicio de sesión. Para esto comenzaremos con el concepto de `Managed Bean`.

Integrando JSF con Maven

¿Qué es un Managed Bean?

Un `Managed Bean` es una clase que sigue la nomenclatura de los Java Beans. El objetivo de estos objetos es controlar el estado de la página web. `JSF` va a administrar automáticamente los managed beans.

Integrando JSF con Maven

Ahora vamos a integrar un bean que represente un usuario del sistema.

```
package com.miguel.proyecto.web;

public class Usuario {

    private String usuario;
    private String contraseña;
    private String confirmacionContraseña;

    public String getUsuario() {
        return usuario;
    }

    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }
```

Integrando JSF con Maven

```
    public String getContraseña() {
        return contraseña;
    }

    public void setContraseña(String contraseña) {
        this.contraseña = contraseña;
    }

    public String getConfirmacionContraseña() {
        return confirmacionContraseña;
    }

    public void setConfirmacionContraseña(String confirmacionContraseña) {
        this.confirmacionContraseña = confirmacionContraseña;
    }

}
```

Integrando JSF con Maven

Agregamos un bean manejado de JSF.

```
package com.miguel.proyecto.web;

import java.util.Locale;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.context.FacesContext;

@ManagedBean
@RequestScoped
public class RegisterController {

private Usuario user = new Usuario();

    public Usuario getUser() {
        return user;
    }

    public void setUser(Usuario user) {
        this.user = user;
    }
```

Integrando JSF con Maven

```
public RegisterController() {
    FacesContext.getCurrentInstance()
                .getViewRoot()
                .setLocale(new Locale("es-Mx"));
}

public String addUser() {
    if (!user.getContraseña()
             .equals(user.getConfirmacionContraseña())) {
        FacesContext.getCurrentInstance()
                    .addMessage(null
                    , new FacesMessage(FacesMessage.SEVERITY_ERROR
          , "Fallo de registro: Las contraseñas deben coincidir", ""));
    } else {
        FacesContext.getCurrentInstance()
                    .addMessage(null,
                     , new FacesMessage(FacesMessage.SEVERITY_INFO,
        "Felicidades, el registro se ha realizado correctamente", ""));
        user = null;
    }
    return null;
}

}
```

Integrando JSF con Maven

Y finalmente la página que queremos visualizar. En este caso un ejemplo sencillo de validación de contraseñas.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui">
  <h:head>
    <title>Ejemplo de JavaServer Faces</title>
   </h:head>
<h:body>

  <h1>Formulario de registro</h1> <br/>
```

Integrando JSF con Maven

```
<h:form id="myForm">
  <table>
    <tr>
      <td>
        <p:messages id="messages" autoUpdate="true"
                    closable="true" />
      </td>
    </tr>
  </table>
  <table>
    <tr>
      <td>Nombre de usuario</td>
      <td><p:inputText
           value="#{registerController.user.usuario}"
           required="true" id="Username" size="10"/></td>
    </tr>
    <tr>
      <td>Contraseña</td>
      <td><p:password
           value="#{registerController.user.contraseña}"
           required="true" feedback="true" id="Password"/></td>
    </tr>
```

Integrando JSF con Maven

```
        <tr>
          <td>Confirmar Contraseña</td>
          <td><p:password
               value="#{registerController.user. confirmacionContraseña}"
              required="true" feedback="true" id="ConfirmPassword"/></td>
        </tr>
        <tr>
          <td colspan="2" align="center">
            <p:commandButton action="#{registerController.addUser}"
                             value="Registrar"/>
            <p:commandButton value="Reset" update="myForm"
                             process="@this">
              <p:resetInput target="myForm" />
            </p:commandButton>
          </td>
        </tr>
      </table>
    </h:form>
  </h:body>
</html>
```

Integrando JSF con Maven

Y terminamos levantando nuestro sitio con:

```
mvn tomcat7:run
```

E ingresamos a la siguiente url.

`http://localhost:`
`8080/mi-primer-aplicacion-web/registro.xhtml`