Sumario

Especificación JSP (Java Server Pages)

Introducción

En lugar de contenidos estáticos que son indiferentes, Java Servlet se introdujo para generar contenido web dinámico que personaliza las solicitudes de los usuarios (por ejemplo, en respuesta a consultas y solicitudes de búsqueda). Sin embargo, es un dolor de cabeza usar un servlet para producir una página HTML presentable (a través de las instrucciones de programación out.prinltn ()). Incluso es peor mantener o modificar esa página HTML producida. Los programadores, que escribieron el servlet, pueden no ser buenos diseñadores gráficos, mientras que un diseñador gráfico no entiende la programación de Java.
JavaServer Pages (JSP) es una tecnología complementaria a java servlet que facilita la mezcla de contenido web dinámico y estático. JSP es la respuesta de Java a las populares páginas Active Server (ASP) de Microsoft o lenguajes como php.
JSP proporciona una forma elegante de mezclar contenidos estáticos y dinámicos. La página principal está escrita en HTML normal, mientras que las etiquetas especiales se proporcionan para insertar piezas de códigos de programación de java. La lógica de programación comercial y la presentación están claramente separadas. Esto permite que los programadores se centren en la lógica del negocio, mientras que el diseñador web se concentre en la presentación.
JSP se basa en servlet. De hecho, veremos más adelante que una página JSP se traduce internamente a un servlet Java. También se explicará más adelante que "Servlet es HTML dentro de Java", mientras que "JSP es Java dentro de HTML" . Lo que no se puede hacer en servlet, no se puede hacer en JSP. JSP hace que la creación y el mantenimiento de páginas HTML dinámicas sea mucho más fácil que el servlet. JSP es más conveniente que servlet para manejar la presentación, no más potente.
JSP está destinado a complementar Servlet, no un reemplazo. En un diseño Modelo-Vista-Control (MVC), los servlets se utilizan para el controlador, lo que implica una compleja lógica de programación. Los JSP se usan para la vista, que se ocupa de la presentación. El modelo podría implementarse utilizando JavaBeans o Enterprise JavaBeans (EJB) que pueden interactuar con una base de datos.
Para comprender JSP, debe comprender Servlet (y HTTP, HTML y Java).

Ventajas de JSP

● Separación de contenidos estáticos y dinámicos: los contenidos dinámicos se generan a través de la lógica de programación y se insertan en la plantilla estática. Esto simplifica enormemente la creación y el mantenimiento de los contenidos web.

● Reutilización de componentes y bibliotecas de etiquetas: los contenidos dinámicos pueden ser proporcionados por componentes reutilizables como JavaBean, Enterprise JavaBean (EJB) y bibliotecas de etiquetas: no es necesario reinventar las ruedas.

● Poder y portabilidad de Java.

Fundamentos de JSP

Comencemos con los conceptos básicos de JSP. Una pregunta común que las personas hacen es "¿Qué es un archivo JSP?" En pocas palabras, es una página HTML con un poco de código de Java en el medio. Es una forma de obtener contenido dinámico generado en tiempo de ejecución en

páginas HTML. Los archivos JSP se procesan en el servidor y el resultado final procesado se incluye en el HTML y luego se devuelve al navegador que inició la solicitud.
Para que los archivos funcionen correctamente, debe colocarlos en la carpeta WebContent, con la extensión .jsp. Ahora es como un sitio web normal, por lo que puede tener tantos archivos JSP como desee e incluso puede colocarlos en directorios secundarios como los archivos HTML, solo que tendrá una extensión .jsp en lugar de .html.

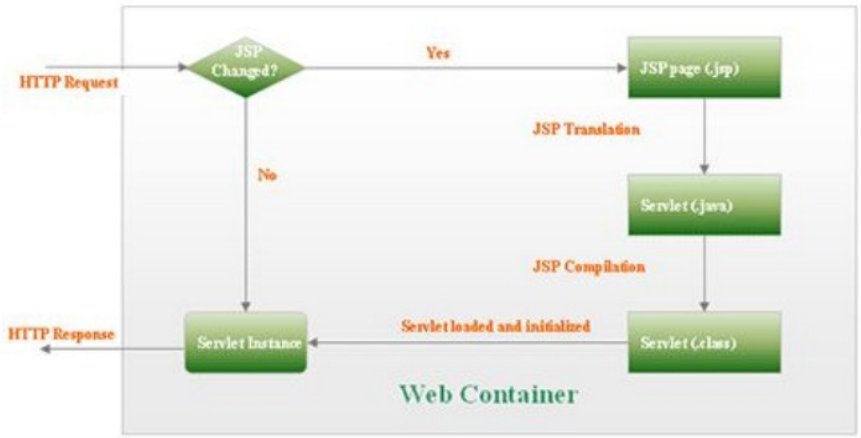
Ejemplo de JSP

```
<html>

<body>

<h2>Hello World!</h2>

The time on server is <%= new java.util.Date() %>

</body>

</html>
```

Para usar el código java usamos "<%". Cuando abre la etiqueta, debe usar un signo igual después del signo de porcentaje. Después de abrir la etiqueta java, creamos un nuevo objeto de fecha que obtendrá la fecha establecida en el servidor y luego, cuando se cierre la etiqueta, el valor devuelto se incluirá en el HTML.

Ciclo de vida de un JSP

JSP nunca envía el contenido directamente al navegador. En su lugar, JSP confía en el proceso de procesamiento inicial del lado del servidor que traduce la página JSP a la clase de página JSP. La clase de Página entonces maneja todas las solicitudes hechas de JSP. El ciclo de vida JSP se describe en la siguiente imagen:



El ciclo de vida JSP se puede dividir en cuatro fases: traducción, inicialización, ejecución y finalización.

Traducción

En la fase de traducción, el motor JSP busca la sintaxis JSP y traduce la página JSP a su clase de implementación de página si la sintaxis es correcta. Esta clase es en realidad un servlet java estándar. Después de eso, el motor JSP compila el archivo fuente en un archivo de clase que está listo para usar.
Si el contenedor recibe la solicitud, verifica los cambios de la página JSP desde la última traducción. Si no se realizaron cambios, simplemente carga el servlet; de lo contrario, el proceso de verificación, traducción y compilación tiene lugar nuevamente.
Debido a que el proceso de compilación lleva tiempo, el motor JSP desea minimizarlo para aumentar el rendimiento del procesamiento de la página.

Inicialización

Después de la fase de traducción, el motor JSP carga el archivo de clase y crea una instancia del servlet para gestionar el procesamiento de la solicitud inicial.
Los motores JSP llamarán a un método jspInit() para inicializar el servlet. El método jspInit() se genera durante la fase de traducción que normalmente se usa para inicializar los parámetros y recursos del nivel de la aplicación. También puede anular este método usando la siguiente declaración:

```
<%!
   public void jspInit(){
      // put your custom code here
   }
%>
```

Ejecución

Después de la fase de inicialización, el contenedor web llama al método _jspService() para manejar la solicitud y devolver una respuesta al cliente. Cada solicitud se maneja es un hilo separado. Tenga en cuenta que todos los scriptlets y expresiones terminan dentro de este método. Las directivas JSP y la declaración se aplican a toda la página, por lo que están fuera de este método.

Finalización

En la fase de finalización, el contenedor web llama al método jspDestroy().
Este método se usa para limpiar la memoria y los recursos. Al igual que el método jspInit(), también puede anular el método jspDestroy() para realizar todas las tareas de limpieza, como liberar los recursos que cargó en la fase de inicialización:

```
<%!
   public void jspDestroy(){
      // put your custom code here
      // to clean up resources
   }
%>
```

JSP Scripting

La creación de scripts JSP le permite insertar código Java en servlet java generado a partir de la página JSP. Estos son los elementos de scripting: comentario, expresión, scriptlet, declaración y lenguaje de expresión.

Comentario JSP

El comentario de JSP es para documentar el código. Además, el comentario JSP se utiliza para señalar algunas partes de la página JSP para que sean más claras y fácil de mantener. A continuación, se ilustra el comentario de JSP dentro de una página JSP:

```
<%-- This is a JSP comment --%>

<%--
   This is a JSP comment can span
   in multiple lines
--%>
```

El comentario de JSP se elimina de la página en la respuesta al navegador web.

Expresión

La expresión es uno de los elementos de scripting más básicos en JSP. La expresión se usa para insertar valor directamente en la salida. La sintaxis de una expresión es la siguiente:

```
<?= expression ?>
```

Se nota que no hay espacio entre <% y =. Por ejemplo, si desea mostrar la fecha y la hora actuales, puede usar una expresión de la siguiente manera:

```
<%= new java.util.Date()%>
```

También puede usar la sintaxis XML de la expresión JSP para lograr el mismo efecto que en el siguiente ejemplo:

```
<jsp:expression>
  Java Expression
</jsp:expression>
```

Scriptlet

Scriptlet es similar a la expresión, excepto por el signo igual "=". Puede insertar cualquier código Java simple dentro de un scriptlet. Debido a la mezcla entre el código Java y HTML, hacer que la página JSP sea difícil de mantener, por lo que scriptlet no se recomienda para desarrollos futuros.

```jsp
<% // any java source code here %>
```

En este ejemplo, mostramos un mensaje de saludo basado en la hora del día usando el scriptlet.

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <title>JSP syntax</title>
    </head>
    <body>
        <%
                // using scriptlet
                java.util.Calendar now = new java.util.GregorianCalendar(
                String tod = "";

                if (now.get(now.HOUR_OF_DAY) < 12) {
                    tod = "Morning!";
                } else if (now.get(now.HOUR_OF_DAY) < 18) {
                    tod = "Afternoon!";
                } else {
                    tod = "Evening!";
                }
        %>

        Good <%=tod%>

    </body>
</html>
```

Declaración

Si se desea definir métodos o campos, se puede usar la declaración JSP. La declaración JSP está rodeada por el signo <%! y%>. Por ejemplo, si desea declarar una variable x, puede usar la declaración JSP de la siguiente manera:

```jsp
<%! int x = 10; %>
```

El punto y coma final es obligatorio. La diferencia entre una variable que usa declaración y una variable se declara usando scriptlet es que una variable declarada usando la etiqueta de declaración es accesible por todos los métodos, mientras que una variable declarada usando scriptlet sólo es accesible para el método _jspservice() del servlet generado del JSP página.
También podemos declarar un método usando la etiqueta de declaración como el siguiente ejemplo:

```jsp
<%!
public boolean isInRange(int x,int min,int max){
        return x >= min && x <= max;
}
%>
```

HTML Form

HTML básicamente es una interfaz gráfica de usuario (GUI) que se presenta para obtener los datos de entrada de los usuarios. Una vez que los usuarios envían el formulario desde el lado del cliente, en el lado del servidor, se deben capturar esos datos para su posterior procesamiento, como la validación de la lógica de negocios, guardar los datos en la base de datos, etc.
Comencemos con un formulario HTML simple para obtener información del usuario.

```
<html>
    <head>
        <title>JSP Form Demo</title>
        <style type="text/css">
            label{ margin-right:20px;}
            input{ margin-top:5px;}
        </style>
    </head>
    <body>
        <form action="handleUserInfo.jsp" method="post">
            <fieldset>
                <legend>User Information</legend>
                <label for="fistName">First Name</label>
                <input type="text" name="firstName" /> <br/>
                <label for="lastName">Last Name</label>
                <input type="text" name="lastName" /> <br/>
                <label for="email">Email</label>
                <input type="text" name="email" /> <br/>
                <input type="submit" value="submit">
            </fieldset>
        </form>
    </body>
</html>
```

El formulario es muy simple y contiene tres campos: nombre, apellido y correo electrónico. Además, contiene el botón de enviar que los usuarios pueden enviar cuando terminan de ingresar los datos.
En la etiqueta de formulario, puede ver que usa el método de publicación para publicar los datos del formulario en el servidor. Necesitamos crear un archivo JSP que sea responsable de obtener y procesar los datos del formulario.
El siguiente es el formulario handleUserInfo.jsp:

```
<html>
    <head>
        <title>JSP Form Demo</title>
    </head>
    <body>
        <%
            String firstName = request.getParameter("firstName");
            String lastName = request.getParameter("lastName");
            String email = request.getParameter("email");
        %>
        <p>Hi <%=firstName%> <%=lastName%>!,
your submitted email is <%=email%>.</p>
    </body>
</html>
```

Usamos el objeto request para obtener los datos del formulario. El método getParameter () del objeto request acepta el nombre del campo de formulario y el valor del campo de retorno. El valor

devuelto del método getParameter () siempre es un tipo texto, por lo tanto, si tiene un campo de formulario que acepte el valor numérico, debe convertirlo.
Si no se encuentra ningún campo de formulario, el método getParameter() devuelve null. Después de obtener los valores del formulario, handleUserInfo.jsp usa esos valores para mostrar un mensaje.
En este ejemplo, se mostró cómo usar el objeto request para obtener los datos de un formulario HTML. El código del formulario HTML y la página JSP que maneja el formulario están en archivos separados.
Al tratar con HTML, tiene varias formas de separar el código para manejar los datos del formulario y el formulario HTML en sí. Estas son las dos formas más comunes de hacerlo:

● La página que contiene el Formulario HTML y la página JSP que maneja el Formulario HTML están separadas.

● Una sola página JSP para mostrar el formulario y manejar el formulario.
Para hacerlo de la segunda manera, necesitamos agregar un campo oculto adicional en el formulario HTML. Cuando el usuario envía los datos del formulario, verificamos el valor de este campo oculto para ver si el formulario se envía o no.

```
<%
    String val = request.getParameter("isSubmitted");
    int isSubmitted = 0;
    if (val != null) {
        isSubmitted = Integer.parseInt(val);
        if (isSubmitted == 1) {
            String firstName = request.getParameter("firstName");
            String lastName = request.getParameter("lastName");
            String email = request.getParameter("email");
            out.println("<p>Hi " +
                        firstName + " " +
                        lastName + "!,
your submitted email is " + email +"</p>");
        }
    }
%>
<% if (isSubmitted == 0) {%>
<form action="index.jsp" method="post">
    <fieldset>
        <legend>User Information</legend>
        <label for="fistName">First Name</label>
        <input type="text" name="firstName" /> <br/>
        <label for="lastName">Last Name</label>
        <input type="text" name="lastName" /> <br/>
        <label for="email">Email</label>
        <input type="text" name="email" /> <br/>
        <input type="hidden" name="isSubmitted" value="1" />
        <input type="submit" value="submit">
    </fieldset>
</form>
<%}%>
```

Lo primero que se puede ver es que la acción del formulario ahora apunta a la página cuyo formulario está incrustado o en la página index.jsp. En segundo lugar, la lógica del procesamiento

de los datos del formulario y la visualización del formulario se encuentran en una sola página JSP. La secuencia de eventos ocurre de la siguiente manera:

1. En la primera vez que se carga la página, el código verifica que el nombre del campo del formulario sea enviado. Porque getParameter devolverá nulo, por lo que la variable isSubmitted se establece en 0 y el código muestra el formulario.
2. Cuando el usuario envía el formulario, hay un campo oculto llamado isSubmitted con el valor 1 disponible en el formulario. Por lo tanto, en estemomento el parámetro es isSubmitted está disponible. El código aún verifica que el parámetro sea enviado y lo encuentre. Luego, el proceso de recopilación de datos e impresión de ellos ocurre normalmente, pero no se genera ningún formulario después de la publicación posterior.

Una de las cosas importantes a tener en cuenta cuando se ocupa del formulario es que: Nunca confíe en los datos enviados desde el lado del cliente. Puede hacer la validación del cliente para la validación simple, como el campo requerido, el correo electrónico está en el formato correcto al usar JavaScript. En el lado del servidor, también puede realizar una validación simple después de recopilar los datos, además de la validación empresarial para validar las restricciones entre los datos que recopiló.

Directivas JSP

Las directivas JSP instruyen al contenedor JSP para que funcione con las páginas JSP. Hay tres directivas en la especificación JSP: página, include y taglib. Las directivas siguen los siguientes formas:

```
<%@ directive attribute1 = "value 1"
              attribute2 = "value 2"
              ...
              %>
```

Directiva de página

La directiva de página le permite definir uno o más de los siguientes atributos:

Import

```
<em>import="package.class"</em>
<em>import="package.class1,...,package.classN"</em>
```

La opción de importación le permite especificar qué clases o paquetes se usan en la página. por ejemplo, puede importar todas las clases en el paquete java.util de la siguiente manera:

```
<%@ page import="java.util.*" %>
```

Content Type

La opción contentType le permite especificar el tipo MIME de la página de salida. El valor predeterminado es text/html. También puede usar otros valores como text/plain para generar formato de texto de la siguiente manera:

```
<%@ page contentType="text/plain" %>
```

Option

Al usar la opción de sesión, le está diciendo al compilador JSP que desea usar la sesión o no. El valor predeterminado del atributo de sesión es verdadero, lo que indica que hay una variable implícita llamada sesión disponible.

Error Page

La opción errorPage le permite indicar la página de error cuando ocurre un error en la página JSP de ejecución actual.

IsErrorPage

La opción isErrorPage indica que la página JSP actual se puede usar como una página de error para otras páginas JSP. Cuando establece isErrorPage como verdadero, el motor JSP crea un objeto de excepción implícito que contiene el objeto Throwable que desencadena la llamada a la página de error. Al usar este objeto de excepción, puede mostrar mensajes de error.

Directiva Include

La directiva Include le permite incluir un archivo (JSP o HTML) cuando el motor JSP traduce la página JSP en un servlet. La sintaxis de la directiva include es la siguiente:

```
<%@ include file="relative url" %>
```

El nombre de archivo que especifique en la directiva include es una URL relativa. Si proporciona un nombre de archivo sin ruta asociada, el compilador JSP siempre asume que el archivo está en el mismo directorio que la página JSP actual.
Normalmente, la directiva include se usa para incluir secciones comunes de una página JSP en una aplicación web o sitio web, como encabezado, barra de navegación, pie de página ... para que esas secciones sean reutilizables en cada página JSP.

Taglib

JSP tiene un conjunto de etiquetas estándar disponibles. JSP también le permite crear nuevas etiquetas personalizadas que se parecen a etiquetas HTML o XML. En este caso, puede usar la directiva taglib para usar su etiqueta personalizada en su página JSP.

```
<%@ taglib uri="http://localhost/jsptutorial/taglib/mytaglib"
   prefix="jsptutorial" %>
```

Uso de Sessiones

El protocolo HTTP es sin estado. Significa que no hay una conexión permanente entre el cliente (navegador web) y el servidor web. Cuando un cliente solicita una página de un servidor web, abre una conexión, recupera la página y cierra la conexión. Los servidores web no saben qué sucede entonces en el lado del cliente.Además, si se realiza otra solicitud de un cliente, el servidor web no asocia la nueva conexión con la conexión que se ha realizado.
Para superar la falta de estado del protocolo HTTP, JSP le proporciona el objeto de sesión implícito que es un objeto HttpSession. El objeto de la sesión reside en el lado del servidor para que pueda guardar datos arbitrarios sobre el cliente y otros datos también en la sesión y más adelante en diferentes solicitudes puede recuperar los datos guardados para su procesamiento. JSP almacena datos en el lado del servidor en el objeto de sesión mediante el uso de una sola clave que el cliente recuerda.

Cómo funciona la sesión

Cuando el servidor crea una nueva sesión, siempre agrega un identificador de sesión en forma de cookie. Cuando el navegador web solicita una página o realiza una solicitud, el navegador web siempre envía una cookie creada por el servidor web en la solicitud. Por lo tanto, en el lado del servidor, el servidor web busca esa cookie y encuentra la sesión correspondiente que coincide con la cookie recibida.
La sesión normalmente dura poco, por lo que la cookie de sesión no se guarda en el disco. La sesión también tiene un tiempo de espera. Cuando se acaba el tiempo, la sesión ya no existe en el lado del servidor. Puede configurar el tiempo de espera de la sesión en el archivo de configuración en el servidor.

Trabajando con Cookies

Una cookie es una pequeña información que se almacena en la computadora del usuario. El servidor web usa una cookie para identificar al usuario en la próxima visita.
Cada vez que el usuario visita un sitio web cuya cookie está habilitada, el servidor web agrega datos adicionales en el encabezado HTTP y responde al navegador web. la próxima vez que un usuario visite el mismo sitio nuevamente, elnavegador web también envía una cookie en el encabezado de solicitud HTTP al servidor web.
El usuario también puede deshabilitar las cookies en el navegador web que admite desactivar la función de cookies como Mozilla Firefox, IE …
Una cookie se almacena en la computadora del usuario como una cadena de nombre / valor. Además, una cookie tiene atributos como dominio, ruta y tiempo de espera.
JSP proporciona API para que pueda trabajar con cookies de manera efectiva a través del objeto de la clase javax.servlet.http.Cookie.

```
<%@page import="javax.servlet.http.Cookie"%>
<html>
    <head>
        <title>JSP Set Cookie</title>
    </head>
    <body>
        <%
            Cookie cookie = new Cookie("ClientID","JSP Guru");
cookie.setMaxAge(3600);
                        response.addCookie(cookie);
        %>
    </body>
</html>
```

JSP Standard Tag Library

JSP está diseñado para la capa de presentación en las aplicaciones web, pero debe contener la lógica o el código dentro de la página para controlar la forma en que presenta los elementos visuales. Desde que se inventó JSP, scriptlet se han utilizado intensamente, por lo tanto, las páginas JSP se vuelven complicadas y difíciles de mantener. El HTML mezclado con el scriptlet JSP y el corchete de apertura y cierre hacen que la página JSP sea incluso difícil de extender. En junio de 2002, se publicó por primera vez la especificación 1.0 de la biblioteca estándar de JavaServer Pages (JSTL). JSTL proporcionó nuevas formas para que el autor de JSP trabaje con diferentes elementos con etiquetas amigables estándar.

La biblioteca de etiquetas estándar JSP se puede dividir en cuatro bibliotecas de etiquetas de la siguiente manera:

1. Core tags
2. Internationalization (i18n) and formatting tags
3. Relational database access tags
4. XML processing tags

Los objetivos de la biblioteca de etiquetas anterior son:
● Simplificar la tarea de escribir páginas JSP proporcionando etiquetas de base amigables XML.
● Proporcionar una lógica reutilizable desde la presentación de la página.
● Hacer que la página JSP sea más fácil de leer y mantener.

Core

Como su nombre lo indica, las etiquetas core proporcionan las acciones de funcionalidad principal a JSP para que las acciones más comunes sean más fáciles de lograr de una manera más efectiva. Las etiquetas principales especifican varias acciones, como mostrar contenido según la condición, manipular colecciones y administrar URL. Al usar las etiquetas centrales, nunca tendrás que escribir una pequeña pieza de scriptlet.

Internationalization (i18n) and formatting tags

Estas etiquetas especifican una serie de acciones para hacer que la aplicación web sea multilingüe. Esas acciones incluyen paquete de recursos, configuraciones regionales y nombres base.

Relational database access tags

Acceder a la base de datos es una de las tareas más importantes de las aplicaciones web. JSTL proporciona una lista de etiquetas estándar para ayudarle a manipular datos como seleccionar, insertar, actualizar y eliminar de las bases de datos relacionales.

XML processing tags

XML se convierte en un estándar de aplicación web empresarial para el intercambio de datos. Manipular XML de manera eficaz, por lo tanto, es muy importante para la mayoría de las

aplicaciones web y, por supuesto, JSTL también proporciona una lista de etiquetas para procesar desde el análisis XML hasta la transformación XML.

Ejemplo de Uso de JSTL

```
<!DOCTYPE html>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql"%>
<html>
    <head>
        <title>Start Page</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <body>
        <h1>Test JSTL</h1>
        <!-- tag out -->
        <c:out value="Hola mundo, soy el tag out!" />

        <!-- tag set -->
        <%-- Setea una variable --%>
        <c:set var="miVariable" scope="request" value="soy el valor" />

        <%-- Imprime el valor miVariable --%>
        <c:out value="miVariable" />

        <P>

            <%-- Imprime el contenido de miVariable --%>
            <c:out value="${miVariable}" />

        <P>


            <%-- Define variables --%>
            <c:set var="vacio" scope="request" value="" />
            <c:set var="miVariable" scope="request" value="100" />


            <c:if test="${ miVariable == '100'}">
                <c:out value="miVariable es igual a 100" />
            </c:if>

            <c:if test="${!(miVariable == '100')}">
                <c:out value="miVariable NO es igual a 100" />
            </c:if>


            <%--
                    Para ejecutar este ejemplo: param.jsp?
saludar=true&nombre=Juan
            --%>
            <c:if test="${param.saludar}">
            <h1>Hola ${param.nombre}!</H1>
            </c:if>

        <jsp:useBean id="now" class="java.util.Date" />
        Date: <fmt:formatDate value="${now}" dateStyle="full" />
    <br>
    <br>
```

```
    Date: <fmt:formatDate value="${now}" pattern="EEE, MMM d, yy" />

    <%--
        Para formatear una fecha se puede utilizar la propiedad 'pattern' del
tag fmt:formatDate
        Ejemplos de uso:

            Date and Time Pattern           Result
            ----------------------------------------------------------------
            "yyyy.MM.dd G 'at' HH:mm:ss z"     2001.07.04 AD at 12:08:56 PDT
            "EEE, MMM d, ''yy"                  Wed, Jul 4, '01
            "h:mm a"                            12:08 PM
            "hh 'o''clock' a, zzzz"             12 o'clock PM, Pacific Daylight
Time
            "K:mm a, z"                         0:08 PM, PDT
            "yyyyy.MMMMM.dd GGG hh:mm aaa"     02001.July.04 AD 12:08 PM
            "EEE, d MMM yyyy HH:mm:ss Z" Wed, 4 Jul 2001 12:08:56 -0700
            "yyMMddHHmmssZ"                     010704120856-0700
            "yyyy-MM-dd'T'HH:mm:ss.SSSZ" 2001-07-04T12:08:56.235-0700
    ---%>

    <%-- Define variables --%>
    <c:set var="categoria" value="C" scope="request" />

    <%-- Utiliza el choose --%>
    <c:choose>
        <c:when test="${categoria == 'A'}">
            <c:out value="Es la categoría A" />
        </c:when>
        <c:when test="${categoria == 'B'}">
            <c:out value="Es la categoría B" />
        </c:when>
        <c:when test="${categoria == 'C'}">
            <c:out value="Es la categoría C" />
        </c:when>
        <c:otherwise>
            <c:out value="No es una categoria" />
        </c:otherwise>
    </c:choose>



    <%-- Itera desde 10 al 20 saltando de a dos --%>
    <c:forEach var="i" begin="10" end="20" step="2">
        <c:out value="${i}"/><BR>
    </c:forEach>

    <HR>


    <%-- Itera desde 10 al 20 saltando index y count--%>
    <c:forEach var="i" begin="10" end="20" varStatus="status">
        indice: <c:out value="${status.index}"/>
        iteracion #: <c:out value="${status.count}"/> <BR>
    </c:forEach>

    <HR>


    <%-- Itera por los items de una lista --%>

    <jsp:useBean id="in" scope="request" class="repositories.Inscripto" />

    <c:forEach var="i" items="${in.all}">
```

```
        <c:out value="${i.id}" /> | <c:out value="${i.nombre}" /> | <c:out
value="${i.apellido}" /><BR>
    </c:forEach>

    <table border="1">
        <thead><th>id</th><th>nombre</th><th>apellido</th></thead>
        <c:forEach var="i" items="${in.all}">
            <tr>
                <td><c:out value="${i.id}" />       </td>
                <td><c:out value="${i.nombre}" />   </td>
                <td><c:out value="${i.apellido}" /> </td>
            </tr>
        </c:forEach>
    </table>


    <%-- JSTL SQL --%>


        <sql:setDataSource
            var="ds"
            driver="com.mysql.cj.jdbc.Driver"
            url="jdbc:mysql://localhost/inscriptos"
            user="root"
            password=""
        />
        <sql:query var="allRows" dataSource="${ds}">
            select id,nombre,apellido,direccion,genero,puesto,experiencia from
inscriptos;
        </sql:query>

         <c:forEach var="in" items="${allRows.rows}">
            <c:out value="${in.id}" />
            <c:out value="${in.nombre}" />
            <c:out value="${in.apellido}" />
            <c:out value="${in.direccion}" />
            <c:out value="${in.genero}" />
            <c:out value="${in.experiencia}" />
        </c:forEach>

    </body>
</html>
```