	INGENIERÍA EN INFORMÁTICA – PLAN 2003/2014 PROGRAMACIÓN DISTRIBUIDA Y COMPONENTES – 9º CUATRIMESTRE	
	APUNTE DE CUSTOM TAG LIBRARIES	VERSIÓN: 1.0 VIGENCIA: 16-04-2018

Custom Tag Libraries

Las librerías de tags personalizados (custom tag libraries) permiten encapsular elementos de funcionalidad compleja, para ser usados en un archivo JSP. Los custom tag libraries contienen uno o más custom tags que los desarrolladores pueden usar para crear contenidos dinámicos en sus páginas.

La funcionalidad de estos custom tags se define en Java, mediante clases que implementan la interface Tag del package javax.servlet.jsp.tagext, normalmente derivando las clases SimpleTagSupport o BodyTagSupport que ya implementan esa interface. Este mecanismo permite a los programadores Java crear elementos de funcionalidad compleja para que sean usados por diseñadores de páginas Web que no tienen conocimientos de programación Java.

Un archivo JSP incluye una custom tag library mediante la directiva taglib, la cual tiene dos atributos:

- **uri**: Especifica el URI (dirección) relativo o absoluto del descriptor de la librería de tags (o tag library descriptor).
- **prefix**: Especifica cual es el prefijo requerido para distinguir a los custom tags de tags propios de la página. Los nombres de prefijos jsp, jsp, java, javax, servlet, sun y sunw son reservados.

Para implementar custom tags, se deben definir tres elementos:

- Una tag-handler class (clase manejadora para el tag) por cada tag, tal que esa clase implemente la funcionalidad del tag.
- Un tag library descriptor (descriptor de librería de tags) que es un archivo con formato XML que provee información sobre la librería de tags y sobre cada tag a la página JSP.
- Un archivo JSP que use el o los custom tags implementados.

Una clase para manejar un custom tag que sólo realice salidas en la página, debe derivar de una clase que haya implementado la interface Tag del paquete **javax.servlet.jsp.tagext**. En este caso, la clase deriva de **SimpleTagSupport**, que cumple esa condición. Si la clase realizará alguna interacción con el cuerpo de la página, entonces la clase debe derivar de **BodyTagSupport** y redefinir algunos métodos para manejar esa interacción.

Veamos un ejemplo sencillo, supongamos que nuestro custom tag mostrará un mensaje en pantalla. Lo primero que debemos definir es la clase manejadora del tag que en nuestro caso se llamará **SaludoTagHandler**, y tiene la siguiente programación:


```
import java.io.IOException;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SaludoTagHandler extends SimpleTagSupport {

    @Override
    public void doTag() throws JspException, IOException {
        super.doTag();
        JspWriter out = this.getJspContext().getOut();
        out.println("Hola Mundo! :)");
    }

}
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003/2014 PROGRAMACIÓN DISTRIBUIDA Y COMPONENTES – 9º CUATRIMESTRE	
	APUNTE DE CUSTOM TAG LIBRARIES	VERSIÓN: 1.0 VIGENCIA: 16-04-2018

En este caso, la clase sólo realiza la salida de un mensaje, por lo que deriva de SimpleTagSupport. Esta clase provee un método doTag(), que es automáticamente invocado por el contenedor JPS cuando este encuentra un tag de apertura o un tag de cierre de un custom tag.

Luego, debemos definir el archivo descriptor de la librería (tag library descriptor o tld) que es un documento XML que contiene elementos requeridos por el contenedor JPS. Para nosotros, el archivo sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
  version="2.0">
  <description>Mis Custom Tag</description>
  <tlib-version>2.1</tlib-version>
  <short-name>customs</short-name>
  <uri>/WEB-INF/tld/customs</uri>
  <tag>
    <name>saludo</name>
    <tag-class>ar.edu.ubp.pdc.tags.SaludoTagHandler</tag-class>
    <body-content>empty</body-content>
  </tag>
</taglib>
```

Un elemento tag incluye algunos atributos: **name** indica el nombre del tag, tal y como será invocado desde la página. El atributo **tag-class** indica el nombre de la clase manejadora del tag, que debe estar incluida en un package y su nombre deberá incluir ese package. El atributo **body-content** indica si el tag tendrá alguna interacción con el cuerpo de la página (vale *empty* si eso no ocurre). Este atributo puede ser *JSP* o también *tagdependent* si ese tag no es vacío.

Finalmente, para utilizarlo dentro de un archivo JSP, debemos agrega la siguiente directiva:

```
<%@ taglib uri="/WEB-INF/tld/customs.tld" prefix="ct" %>
```

En esta definición, se usa el atributo **uri** para indicarle a la página que la dirección del archivo descriptor de la librería se llama *customs.tld*, y que ese archivo está en la carpeta */WEB-INF/tld* del sitio Web. El atributo **prefix** se usa para indicar un alias con el cual usar los tags definidos. En este caso, el alias es *ct*.

Luego, en la sección body de la página se activa un tag de esa librería llamado **saludo**. El efecto de ese tag, es simplemente mostrar en la página el mensaje "¡Hola Mundo! :)". La directiva de activación sería:


```
<ct:saludo />
```

Alternativamente, se podría usar el tag de cierre, así:

```
<ct:saludo></ct:saludo>
```

Notar que la etiqueta de activación usa el prefijo de la librería que contiene al tag para referirse a ese tag. El formato incluye el prefijo, luego dos puntos, y luego el nombre del tag.

Al activar un custom tag se puede enviar algún parámetro al mismo para configurar su funcionalidad. Como podría ser:

	INGENIERÍA EN INFORMÁTICA – PLAN 2003/2014 PROGRAMACIÓN DISTRIBUIDA Y COMPONENTES – 9º CUATRIMESTRE	
	APUNTE DE CUSTOM TAG LIBRARIES	VERSIÓN: 1.0 VIGENCIA: 16-04-2018

```
<ct:mensaje nombre="Mariela" apellido="Pastarini" />
```

La clase manejadora del tag que tome los parámetros declara un atributo con el nombre de cada uno de ellos, a la vez de proveer métodos accesores (tipo get y set) para tomar el valor de esos atributos o cambiarlos. El contenedor del JSP invoca automáticamente a esos métodos accesores para configurar el valor de cada atributo. En nuestro caso, la clase sería:

```
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class MensajeTagHandler extends SimpleTagSupport {

    private String apellido;
    private String nombre;

    @Override
    public void doTag() throws JspException, IOException {
        super.doTag();

        String nom = (this.nombre == null ? "" : this.nombre);
        String ape = (this.apellido == null ? "" : this.apellido);


        JspWriter out = this.getJspContext().getOut();
        out.println("Este es un mensaje para \"" + (nom.equals("") &&
ape.equals("") ? "alguien" : (nom + " " + ape).trim()) + "\"");
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

Por su parte, el archivo descriptor de la librería debería contener la siguiente declaración de nuestro nuevo custom tag:

```
<tag>
    <name>mensaje</name>
    <tag-class>ar.edu.ubp.pdc.tags.MensajeTagHandler</tag-class>
    <body-content>empty</body-content>
    <attribute>
        <name>nombre</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
        <type>java.lang.String</type>
    </attribute>
    <attribute>
        <name>apellido</name>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003/2014 PROGRAMACIÓN DISTRIBUIDA Y COMPONENTES – 9º CUATRIMESTRE	
	APUNTE DE CUSTOM TAG LIBRARIES	VERSIÓN: 1.0 VIGENCIA: 16-04-2018

```

        <type>java.lang.String</type>
    </attribute>
</tag>

```

Como puede observarse, se agregaron subelementos **<attribute>** que describen los atributos nombre y apellido. En cada subelemento, **name** es el nombre del atributo. El indicador **required** especifica si ese atributo es obligatorio o no al invocar el custom tag. Y el elemento **rtexprvalue** indica si el valor del atributo puede ser el resultado de una expresión JPS evaluada en tiempo de ejecución (true) o debe ser una cadena literal (false). Finalmente, el elemento **type** determina el tipo de dato del atributo que se define.

Los custom tags son particularmente poderosos para realizar procesamiento de los elementos del cuerpo de la página. Cuando un custom tag interactúa con los elementos del cuerpo, se necesitan métodos adicionales que realicen esa interacción. Esos métodos son definidos por la clase **BodyTagSupport**.

Supongamos que implementamos un custom que procese tareas de una agenda, como podemos tener muchas tareas mostraremos las mismas en una tabla. El código de la página JSP que utiliza el custom tag sería:

```


<table>
    <colgroup>
        <col width="250px">
        <col width="50px">
        <col width="50px">
        <col width="250px">
    </colgroup>
    <thead>
        <tr>
            <th align="left">Tarea</th>
            <th>Fecha</th>
            <th>Prioridad</th>
            <th align="left">Responsable</th>
        </tr>
    </thead>
    <tbody>
        <ct:agenda>
            <tr>
                <td><%= tarea %></td>
                <td align="center"><nobr><%= fecha %></nobr></td>
                <td><%= prioridad %></td>
                <td><%= responsable %></td>
            </tr>
        </ct:agenda>
    </tbody>
</table>

```

Observemos que el código está utilizando variables resaltadas en negrita, las cuales no están definidas en la página JSP sino que fueron declaradas en la clase manejadora del custom tag y luego colocadas en el contexto de la página. También, es importante destacar que no se incluye ningún ciclo, aunque los datos son procesados en un bucle dentro del custom y desde allí se envían a la tabla.

La clase manejadora es *AgendaTagHandler* y tiene el siguiente código:

```
import java.io.IOException;
```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003/2014 PROGRAMACIÓN DISTRIBUIDA Y COMPONENTES – 9º CUATRIMESTRE	
	APUNTE DE CUSTOM TAG LIBRARIES	VERSIÓN: 1.0 VIGENCIA: 16-04-2018

```

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.LinkedList;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.BodyTagSupport;

import ar.edu.ubp.pdc.classes.Prioridad;
import ar.edu.ubp.pdc.classes.Tarea;

public class AgendaTagHandler extends BodyTagSupport {

    private static final long serialVersionUID = 3154148448049573092L;
    private LinkedList<Tarea> agenda;
    private Iterator<Tarea> iterator;

    public AgendaTagHandler() throws JspException {
        super();

        try {
            SimpleDateFormat f = new SimpleDateFormat("dd-MM-yyyy");


            this.agenda = new LinkedList<Tarea>();
            this.agenda.add(new Tarea("Cumpleaños de Pili", f.parse("07-03-2018"),
Prioridad.ALTA));
            this.agenda.add(new Tarea("Cumpleaños de Silvana", f.parse("16-05-2018"),
Prioridad.ALTA));
            this.agenda.add(new Tarea("Ir a comprar regalo para Silvana...", f.parse("07-
05-2018"), Prioridad.MEDIA, "Mariela"));
            this.agenda.add(new Tarea("Clase de PDC", f.parse("18-04-2018"),
Prioridad.ALTA, "Mariela"));

            Collections.sort(this.agenda, new Comparator<Tarea>() {
                @Override
                public int compare(Tarea o1, Tarea o2) {
                    return o1.getFechaTarea().compareTo(o2.getFechaTarea());
                }
            });

            this.iterator = this.agenda.iterator();
        }
        catch (ParseException ex) {
            throw new JspException(ex);
        }
    }

    @Override
    public int doStartTag() throws JspException {
        if(this.getProximaTarea()) {
            return EVAL_BODY_BUFFERED;
        }
        return SKIP_BODY;
    }

```

	INGENIERÍA EN INFORMÁTICA – PLAN 2003/2014 PROGRAMACIÓN DISTRIBUIDA Y COMPONENTES – 9º CUATRIMESTRE	
	APUNTE DE CUSTOM TAG LIBRARIES	VERSIÓN: 1.0 VIGENCIA: 16-04-2018

```

    }

    @Override
    public int doAfterBody() throws JspException {
        JspWriter out = this.bodyContent.getEnclosingWriter();
        try {
            this.bodyContent.writeOut(out);
            this.bodyContent.clearBody();
            if(this.getProximaTarea()) {
                return EVAL_BODY_AGAIN;
            }
            return SKIP_BODY;
        }
        catch (IOException ex) {
            throw new JspException(ex);
        }
    }

    @Override
    public int doEndTag() throws JspException {
        return super.doEndTag();
    }

    private boolean getProximaTarea() {
        if(this.iterator.hasNext()) {
            DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT,
this.pageContext.getRequest().getLocale());
            Tarea tarea = Tarea.class.cast(this.iterator.next());


            this.pageContext.setAttribute("tarea", tarea.getNomTarea());
            this.pageContext.setAttribute("fecha",
df.format(tarea.getFechaTarea()));
            this.pageContext.setAttribute("prioridad", tarea.getPrioridad().name());
            this.pageContext.setAttribute("responsable", tarea.getNomResponsable() == null
? "" : tarea.getNomResponsable());
            return true; //EVAL_BODY_TAG
        }
        return false; //SKIP_BODY
    }
}

```

La clase extiende a **BodyTagSupport** ya que el tag va a interactuar con el cuerpo de la página. Declara atributos para manejar la lista de tareas, además de un iterador para recorrerla. El constructor de la clase crea el *LinkedList* y lo inicializa añadiendo a él un par de objetos de la clase Tarea. Esta lista simula ser una "base de datos" que la clase accede para armar la tabla a pedido del usuario.

La clase BodyTagSupport provee varios métodos:

- **doStartTag()**: se invoca cuando el motor JSP encuentra el tag de activación, pero después de haber inicializado los atributos de la clase.
- **doEndTag()**: se invoca cuando el motor JSP finaliza de procesar el tag.
- **doAfterBody()**: se invoca cuando el motor JSP termina de procesar el cuerpo de contenidos del tag.

	INGENIERÍA EN INFORMÁTICA – PLAN 2003/2014 PROGRAMACIÓN DISTRIBUIDA Y COMPONENTES – 9º CUATRIMESTRE	
	APUNTE DE CUSTOM TAG LIBRARIES	VERSIÓN: 1.0 VIGENCIA: 16-04-2018

En nuestro caso, el método *doStartTag()* verifica si la lista contiene elementos, y en ese caso retorna la constante **EVAL_BODY_BUFFERED**, que indica al motor que el cuerpo del tag debe ser procesado. Si la lista estuviera vacía, retorna **SKIP_BODY** para indicar que no debe procesar el cuerpo.

Luego se activa el método *doAfterBody()*, que en esencia verifica si la lista sigue teniendo elementos (luego de haber procesado el primero con *doStartTag()*), y en caso de ser así invoca al método *getProximaTarea()* que es privada a la clase. Ese método toma el siguiente elemento de la lista usando el iterador, y envía los valores de sus atributos al contexto de la página para que esta los despliegue en la tabla. El método *doAfterBody()* vuelve a invocarse, una y otra vez, hasta que no queden elementos en la lista, y en ese momento se activa el método *doEndTag()*.

Sin embargo, resta un detalle: el contenedor JSP no puede crear variables en el contexto de la página, a menos que conozca los nombres y los tipos de esas variables. Por lo tanto, las variables enviadas por el método *getProximaTarea()* a la página no serán reconocidas hasta que se le provea al contenedor esa información. En nuestro caso, se usa una clase con el mismo nombre que la manejadora del tag, pero terminada con **ExtraInfo** para proveer esa información. Esa clase debe derivar de la clase **TagExtraInfo**. El contenedor JSP usa la información provista por una subclase de *TagExtraInfo* para determinar qué variable debería crear o usar en el contexto de la página. Para indicar la información, se sobreescribe el método *getVariableInfo()*, el cual retorna un vector de objetos de la clase *VariableInfo*.

El constructor de la clase *VariableInfo* recibe cuatro parámetros: el nombre de la variable, la clase o tipo de la misma, un booleano que indica si la variable debería ser creada o no por el contenedor, y una constante que indica el ámbito de la variable en la JSP que son tres posibles:

- **NESTED**: indica que la variable vale solo en el cuerpo del tag.
- **AT_BEGIN**: indica que la variable vale en cualquier lugar de la página JSP luego de haber encontrado el tag de activación.
- **AT_END**: indica que la variable puede ser usada en cualquier parte.

En el archivo tld debe incluirse un atributo más en el tag en este caso, se trata de **tei-class** para indicar cuál es la clase de *ExtraInfo* que brinda esa información al JSP. El código sería:

```
<tag>
  <name>agenda</name>
  <tag-class>ar.edu.ubp.pdc.tags.AgendaTagHandler</tag-class>
  <tei-class>ar.edu.ubp.pdc.tags.AgendaTagHandlerExtraInfo</tei-class>
  <body-content>JSP</body-content>
</tag>
```