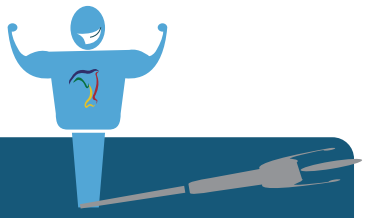
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

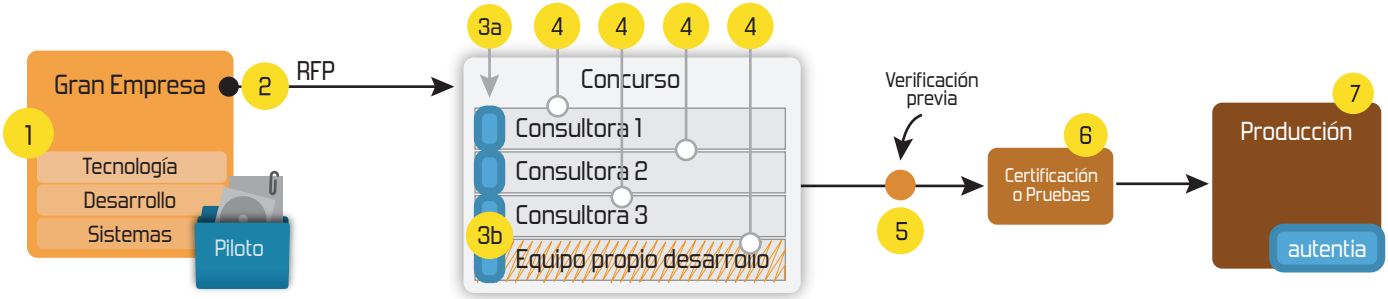
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por:
Roberto Canales Mora 2003-2004.

Una vez escuche esta frase:

Cuando todo esta bajo control, es que no vamos suficientemente deprisa

La mayoría de las veces construimos estos tutoriales al mismo tiempo que recordamos como hacer cosas o probamos nuevas tecnologías, poniendo más celo en el contenido que en la forma (cosa que veo que perturba a muchos de los lectores por lo que os pido disculpas). Es cuestión de criterios .... aunque trataremos de mejorar.

Espero que seáis comprensivos y generosos ayudando a encontrar las erratas, faltas de ortografía, enlaces rotos u otros posibles errores en estas páginas. Escribidme siempre que encontréis algun error y pronto ya no habrá .... y os estaremos francamente agradecidos. Este es tu Web .... colabora libremente ...

Roberto Canales

Descargar este documento en formato PDF dobleclick.pdf

Control de Doble Click en JSP y Struts

Cuando realizamos una aplicación Web, hay un problema bastante común y se produce cuando el usuario realiza una transacción:

- Hace un doble click muy rápido sobre un botón invocando doblemente transacciones
- Se pone nervioso y, antes de que termine, la vuelve a invocar.
- Se produce un error y no está seguro de que las transacciones se hayan confirmado por lo que lo reintenta para ver que le dice el sistema..
- Pulsa el botón volver, refrescar y avanzar con lo que reenvía una misma transacción dos veces.

Una de las soluciones, que habitualmente se utiliza, es escribir un pequeño código JavaScript que nos permita evitar la situación (al menos parcialmente).

El primer problema de esta aproximación es conceptual: Jamás me debo fiar de las validaciones realizadas en el cliente Web (aparte de la parcialidad de la solución) porque:

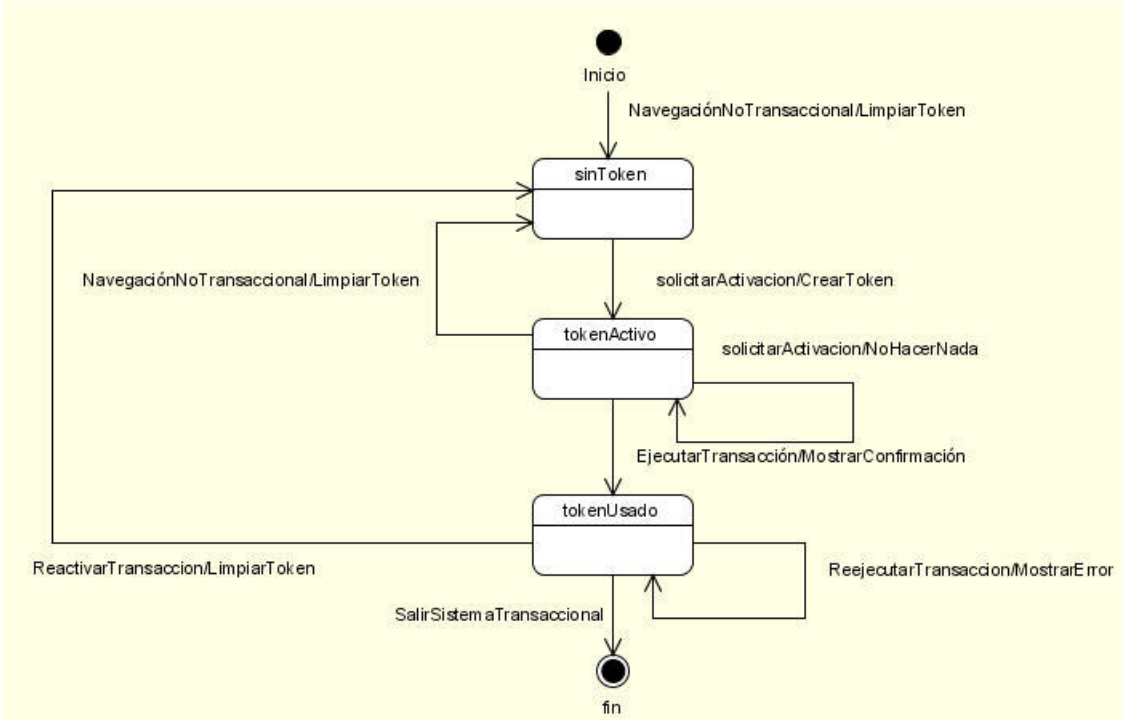
- Son fácilmente evitables.
- JavaScript depende de la versión del navegador (y sistema operativo) que complica la mantenibilidad.
- JavaScript puede desactivarse.
- Podemos hacer un programas que automatice una petición HTTP sin que exista un navegador (y empezad a pensar en WebServices)

Las validaciones importantes hay que controlarlas siempre en el servidor (aunque las complementemos en el cliente para evitar flujos innecesarios de información).

Tenemos que entender los posibles estados que podemos tener el problema:

1. Un usuario llega a un punto que donde se autoriza una nueva transacción (se activa un ticket o token)
2. El usuario ejecuta una vez esa transacción y se deshabilita el token
3. El usuario para poder ejecutar esa misma transacción de nuevo tiene que llegar a un punto que le vuelva a habilitar el token

El párrafo anterior está escrito en un lenguaje poco técnico lo que provoca ambigüedades . Podríamos tratar de utilizar UML para concretar un poquito más los posibles estados y tratar de plasmar en el futuro las posibles variaciones.



Este problema lo podríamos tener tanto en una aplicación con interfaz tipo Windows o en una aplicación Web ..... Lo lógico sería tratar de construir una librería que pudiera se invocada por distintos tipos de aplicaciones.

Nosotros no vamos a ir tan allá pero trataremos de dar el primer paso en la solución creando unas librerías de etiquetas básicas para resolverlo en aplicaciones Web/JSP.

Construiremos un pequeño ejemplo que veremos elemento a elemento.

El formulario

Partimos de un formulario donde mandaremos al servidor un campo oculto. En este campo se almacena un valor que solo es utilizable una vez. El encargado de generar y validar este parámetro es siempre el servidor y lo hacemos a través de una etiqueta. Pasamos como parámetro el nombre de la transacción (**ejemplo**) para poder tener tantos tokens activos como deseemos.

```
<controltokens:generaToken nombreTransaccion="ejemplo"/>
```

```
<%@page contentType="text/html"%>
<%@taglib uri="/roberto/tokens" prefix="controltokens" %>

<html>
<head><title>JSP Page</title></head>
<body>

<center>
<form action='./controldobleclick.jsp'>
Nombre <input type="text" value="">
<input type="HIDDEN" name="tokenEnJSP"
value='<controltokens:generaToken nombreTransaccion="ejemplo"/>'>
<br><input type="submit" value="Pulsar">
</form>

<a href="./anulaToken.jsp">Pulsame para limpiar el Token</a>

</center>
</body>
</html>
```

Podemos ver el aspecto de la página

Y el código que llegaría a nuestro navegador. Hemos elegido como token la fecha actual pero podría ser cualquier cosa

```
<html>
<head><title>JSP Page</title></head>
<body>

<center>
<form action='./controldobleclick.jsp'>
Nombre <input type="text" value="">
<input type="HIDDEN" name="tokenEnJSP"
value='Thu Nov 25 23:15:59 CET 2004'>
<br><input type="submit" value="Pulsar">
</form>

<a href="./anulaToken.jsp">Pulsame para limpiar el Token</a>

</center>
</body>
</html>
```

La rutina transaccional

Cuando ejecutemos la transacción debemos verificar que el toquen es válido. Usamos otro JSP.

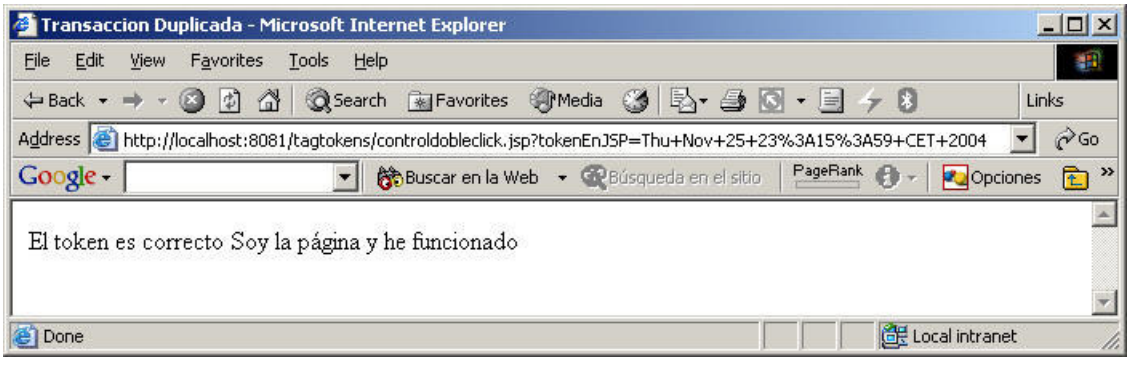
```
<controltokens:compruebaToken nombreTransaccion="ejemplo"/>
```

Si no es así, generamos una excepción y redirigimos a una página de error

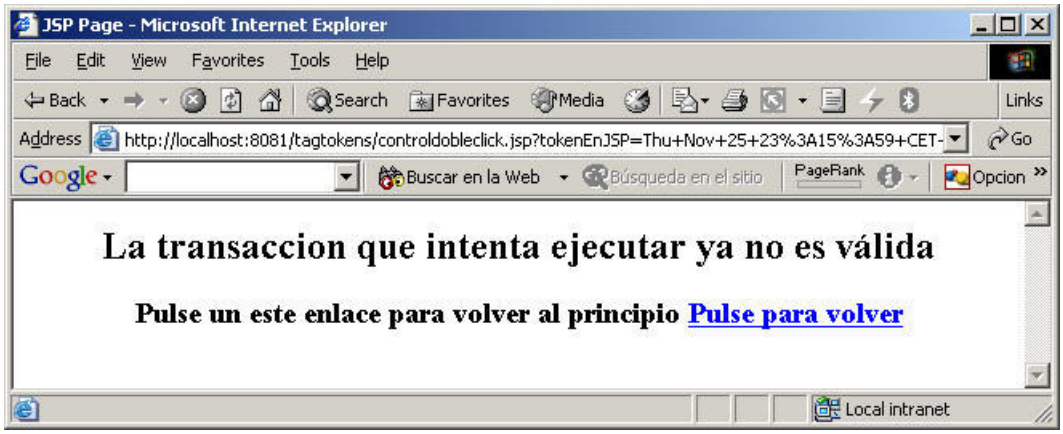
```
errorPage="/pages/transaccionDuplicada.jsp"
```

```
<%@page contentType="text/html" errorPage="/pages/transaccionDuplicada.jsp" %>
<%@taglib uri="/roberto/tokens" prefix="controltokens" %>

<html>
<head><title>Transaccion Duplicada</title></head>
<body>

<controltokens:compruebaToken nombreTransaccion="ejemplo"/>

Soy la página y he funcionado

 ----- Aquí debe ir el código de la transacción ----

</body>
</html>
```

Si la ejecutamos una vez



Si ejecutamos la segunda vez (con refrescar, volver y enviar, volver-refrescar y enviar).



Limpiar el token

En el caso de querer volver a ejecutar la transacción deberíamos pasar por una página que rehabilitase el token.

```
<controltokens:anulaToken nombreTransaccion="ejemplo"/>
```

```
<%@page contentType="text/html" errorPage="/pages/transaccionDuplicada.jsp" %>
<%@taglib uri="/roberto/tokens" prefix="controltokens" %>

<html>
<head><title>Limpieza de Token</title></head>
<body>

<controltokens:anulaToken nombreTransaccion="ejemplo"/>

<a href="./formulariobasico.jsp">Pulsame para ir al formulario</a>

</body>
</html>
```

Librería de TAGS

Ahora solo tenemos que revisar el tutorial donde os mostrábamos como crear paso a paso una etiqueta y analizar el código particular.

El tag de generación de token

```
package tokens;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.BodyContent;
import javax.servlet.jsp.tagext.BodyTag;
import javax.servlet.jsp.tagext.BodyTagSupport;
import javax.servlet.jsp.tagext.IterationTag;
import javax.servlet.jsp.tagext.Tag;
import javax.servlet.jsp.tagext.TagSupport;

import java.util.*;

/**
* Generated tag class.
*/
public class GeneraTokenTag extends TagSupport {

/** property declaration for tag attribute: nombreTransaccion.
*
*/
private String nombreTransaccion;

public GeneraTokenTag() {
super();
}
//////////////////////////////////////////////////////////////////
/// ///
/// User methods. ///
/// ///
/// Modify these methods to customize your tag handler. ///
/// ///
//////////////////////////////////////////////////////////////////

void depura(String mensaje)
{
System.out.println("GeneraTokenTag: " + mensaje);
}
//
```

```
// methods called from doStartTag()
//
/**
*
* Fill in this method to perform other operations from doStartTag().
*
*/
public void otherDoStartTagOperations() {

//
// TODO: code that performs other operations in doStartTag
// should be placed here.
// It will be called after initializing variables,
// finding the parent, setting IDREFs, etc, and
// before calling theBodyShouldBeEvaluated().
//
// For example, to print something out to the JSP, use the following:
//
try {

depura("Cogemos el output");
JspWriter out = pageContext.getOut();

depura("Recuperamos el valor de la sesion");
// recuperamos el valor de la sesion
Object tokenActual = pageContext.getSession().getAttribute(nombreTransaccion);

String texto = null;

if (tokenActual == null)
{
depura("Generamos el token porque no esta en memoria") ;

// el token no esta en memoria
Date fecha = new Date();
texto = fecha.toString();
pageContext.getSession().setAttribute(nombreTransaccion,texto);
}
else
{
depura("El token existe");

// volvemos a poner el valor en memoria

if(tokenActual.toString().compareTo("nulo")==0)
{
texto = "incorrecto";
}
else
{
texto = tokenActual.toString();
}
}

// generamos la respuesta
out.print(texto);
out.flush();

}
catch (java.io.IOException ex) {
// do something
}
}

/**
*
* Fill in this method to determine if the tag body should be evaluated
* Called from doStartTag().
*
*/
public boolean theBodyShouldBeEvaluated() {

//
// TODO: code that determines whether the body should be
// evaluated should be placed here.
// Called from the doStartTag() method.
//
return true;
}

//
// methods called from doEndTag()
//
/**
*
* Fill in this method to perform other operations from doEndTag().
*
*/
public void otherDoEndTagOperations() {

//
// TODO: code that performs other operations in doEndTag
// should be placed here.
// It will be called after initializing variables,
// finding the parent, setting IDREFs, etc, and
// before calling shouldEvaluateRestOfPageAfterEndTag().
```

```
//
}

/**
*
* Fill in this method to determine if the rest of the JSP page
* should be generated after this tag is finished.
* Called from doEndTag().
*
*/
public boolean shouldEvaluateRestOfPageAfterEndTag() {

//
// TODO: code that determines whether the rest of the page
// should be evaluated after the tag is processed
// should be placed here.
// Called from the doEndTag() method.
//
return true;
}

////////////////////////////////////////////////////////////
/// ///
/// Tag Handler interface methods. ///
/// ///
/// Do not modify these methods; instead, modify the ///
/// methods that they call. ///
/// ///
////////////////////////////////////////////////////////////
/** .
*
* This method is called when the JSP engine encounters the start tag,
* after the attributes are processed.
* Scripting variables (if any) have their values set here.
* @return EVAL_BODY_INCLUDE if the JSP engine

should evaluate the tag body, otherwise return SKIP_BODY.
* This method is automatically generated. Do not modify this method.
* Instead, modify the methods that this method calls.
*
*
*/
public int doStartTag() throws JspException, JspException {
otherDoStartTagOperations();

if (theBodyShouldBeEvaluated()) {
return EVAL_BODY_INCLUDE;
} else {
return SKIP_BODY;
}
}

/** .
*
*
* This method is called after the JSP engine finished processing the tag.
* @return EVAL_PAGE if the JSP engine should continue evaluating

the JSP page, otherwise return SKIP_PAGE.
* This method is automatically generated. Do not modify this method.
* Instead, modify the methods that this method calls.
*
*
*/
public int doEndTag() throws JspException, JspException {
otherDoEndTagOperations();

if (shouldEvaluateRestOfPageAfterEndTag()) {
return EVAL_PAGE;
} else {
return SKIP_PAGE;
}
}

public String getNombreTransaccion() {
return nombreTransaccion;
}

public void setNombreTransaccion(String value) {
nombreTransaccion = value;
}

}
```

El tag de Verificación/desactivación

```
package tokens;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
```

```
import javax.servlet.jsp.tagext.BodyContent;
import javax.servlet.jsp.tagext.BodyTag;
import javax.servlet.jsp.tagext.BodyTagSupport;
import javax.servlet.jsp.tagext.IterationTag;
import javax.servlet.jsp.tagext.Tag;
import javax.servlet.jsp.tagext.TagSupport;

import java.util.*;


/**
* Generated tag class.
*/
public class CompruebaTokenTag extends TagSupport {

/** property declaration for tag attribute: nombreTransaccion.
*
*/
private String nombreTransaccion;

public CompruebaTokenTag() {
super();
}


////////////////////////////////////////////////////////////////
/// ///
/// User methods. ///
/// ///
/// Modify these methods to customize your tag handler. ///
/// ///
////////////////////////////////////////////////////////////////


void depura(String mensaje)
{
System.out.println("CompruebaTokenTag: " + mensaje);

}




//
// methods called from doStartTag()
//
/**
*
* Fill in this method to perform other operations from doStartTag().
*
*/
public void otherDoStartTagOperations() throws JspException {

//
// TODO: code that performs other operations in doStartTag
// should be placed here.
// It will be called after initializing variables,
// finding the parent, setting IDREFs, etc, and
// before calling theBodyShouldBeEvaluated().
//
// For example, to print something out to the JSP, use the following:
//
try {
JspWriter out = pageContext.getOut();

// recuperamos el valor de la sesion
Object tokenActual = pageContext.getSession().getAttribute(nombreTransaccion);
String texto = null;

if (tokenActual != null)
{
depura("El token no es nulo");

// el token no esta en memoria
Object tokenPasadoEnJSP = pageContext.getRequest().getParameter("tokenEnJSP");

if(tokenPasadoEnJSP == null)
{
depura("El token es nulo");
throw new JspException("El token no se ha pasado en el JSP como -tokenEnJSP- ");
}

if (tokenPasadoEnJSP.toString().compareTo(tokenActual.toString()) != 0 )
{
depura("El token no coincide con el parámetro");
throw new JspException("El token no coincide con el parámetro -tokenEnJSP- ");
}

depura("El token coincide y ponemos un nuevo valor");

pageContext.getSession().setAttribute(nombreTransaccion,"nulo");
}
else
{
// volvemos a poner el valor en memoria
depura("No hay token en sesion");
```

```
throw new JspException("No hay token en sesion: " + nombreTransaccion);
}

// generamos la respuesta
out.print("El token es correcto");
out.flush();

}
catch (java.io.IOException ex) {
// do something
}

}

/**
*
* Fill in this method to determine if the tag body should be evaluated
* Called from doStartTag().
*
*/
public boolean theBodyShouldBeEvaluated() {

//
// TODO: code that determines whether the body should be
// evaluated should be placed here.
// Called from the doStartTag() method.
//
return true;

}


//
// methods called from doEndTag()
//
/**
*
* Fill in this method to perform other operations from doEndTag().
*
*/
public void otherDoEndTagOperations() {

//
// TODO: code that performs other operations in doEndTag
// should be placed here.
// It will be called after initializing variables,
// finding the parent, setting IDREFs, etc, and
// before calling shouldEvaluateRestOfPageAfterEndTag().
//


}

/**
*
* Fill in this method to determine if the rest of the JSP page
* should be generated after this tag is finished.
* Called from doEndTag().
*
*/
public boolean shouldEvaluateRestOfPageAfterEndTag() {

//
// TODO: code that determines whether the rest of the page
// should be evaluated after the tag is processed
// should be placed here.
// Called from the doEndTag() method.
//
return true;

}


//////////////////////////////////////////////////////////////////
/// ///
/// Tag Handler interface methods. ///
/// ///
/// Do not modify these methods; instead, modify the ///
/// methods that they call. ///
/// ///
//////////////////////////////////////////////////////////////////


/** .
*
* This method is called when the JSP engine encounters the start tag,
* after the attributes are processed.
* Scripting variables (if any) have their values set here.
* @return EVAL_BODY_INCLUDE if the JSP engine should evaluate the tag body,

otherwise return SKIP_BODY.
* This method is automatically generated. Do not modify this method.
* Instead, modify the methods that this method calls.
*
*
*/
```

```
public int doStartTag() throws JspException {
depura("Llegamos a la etiqueta");
otherDoStartTagOperations();

if (theBodyShouldBeEvaluated()) {
return EVAL_BODY_INCLUDE;
} else {
return SKIP_BODY;
}
}

/** .
*
*
* This method is called after the JSP engine finished processing the tag.
* @return EVAL_PAGE if the JSP engine should continue evaluating the JSP page,


 otherwise return SKIP_PAGE.
* This method is automatically generated. Do not modify this method.
* Instead, modify the methods that this method calls.
*
*
*/
public int doEndTag() throws JspException {
otherDoEndTagOperations();

if (shouldEvaluateRestOfPageAfterEndTag()) {
return EVAL_PAGE;
} else {
return SKIP_PAGE;
}
}

public String getNombreTransaccion() {
return nombreTransaccion;
}

public void setNombreTransaccion(String value) {
nombreTransaccion = value;
}

}
```

El tag de reactivación

```
package tokens;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.BodyContent;
import javax.servlet.jsp.tagext.BodyTag;
import javax.servlet.jsp.tagext.BodyTagSupport;
import javax.servlet.jsp.tagext.IterationTag;
import javax.servlet.jsp.tagext.Tag;
import javax.servlet.jsp.tagext.TagSupport;

/**
* Generated tag class.
*/
public class AnulaTokenTag extends TagSupport {

/** property declaration for tag attribute: nombreTransaccion.
*
*/
private String nombreTransaccion;

public AnulaTokenTag() {
super();
}


////////////////////////////////////////////////////////////////
/// ///
/// User methods. ///
/// ///
/// Modify these methods to customize your tag handler. ///
/// ///
////////////////////////////////////////////////////////////////



void depura(String mensaje)
{
System.out.println("AnulaTokenTag: " + mensaje);

}


//
// methods called from doStartTag()
//
```

```
/**
*
* Fill in this method to perform other operations from doStartTag().
*
*/
public void otherDoStartTagOperations() {

//
// TODO: code that performs other operations in doStartTag
// should be placed here.
// It will be called after initializing variables,
// finding the parent, setting IDREFs, etc, and
// before calling theBodyShouldBeEvaluated().
//
// For example, to print something out to the JSP, use the following:
//
try {
JspWriter out = pageContext.getOut();
out.println("Borramos el token " + nombreTransaccion);

pageContext.getSession().removeAttribute(nombreTransaccion);

// generamos la respuesta
out.print("Token anulado");
out.flush();

}
catch (java.io.IOException ex)
{
// do something
}
}

/**
*
* Fill in this method to determine if the tag body should be evaluated
* Called from doStartTag().
*
*/
public boolean theBodyShouldBeEvaluated() {

//
// TODO: code that determines whether the body should be
// evaluated should be placed here.
// Called from the doStartTag() method.
//
return true;

}


//
// methods called from doEndTag()
//
/**
*
* Fill in this method to perform other operations from doEndTag().
*
*/
public void otherDoEndTagOperations() {

//
// TODO: code that performs other operations in doEndTag
// should be placed here.
// It will be called after initializing variables,
// finding the parent, setting IDREFs, etc, and
// before calling shouldEvaluateRestOfPageAfterEndTag().
//


}

/**
*
* Fill in this method to determine if the rest of the JSP page
* should be generated after this tag is finished.
* Called from doEndTag().
*
*/
public boolean shouldEvaluateRestOfPageAfterEndTag() {

//
// TODO: code that determines whether the rest of the page
// should be evaluated after the tag is processed
// should be placed here.
// Called from the doEndTag() method.
//
return true;

}


//////////////////////////////////////////////////////////////
/// ///
/// Tag Handler interface methods. ///
/// ///
```

```
/// Do not modify these methods; instead, modify the ///
/// methods that they call. ///
/// ///
//////////////////////////////////////////////////////////////


/** .
*
* This method is called when the JSP engine encounters the start tag,
* after the attributes are processed.
* Scripting variables (if any) have their values set here.
* @return EVAL_BODY_INCLUDE if the JSP engine should evaluate the tag body,

otherwise return SKIP_BODY.
* This method is automatically generated. Do not modify this method.
* Instead, modify the methods that this method calls.
*
*
*/
public int doStartTag() throws JspException, JspException {
otherDoStartTagOperations();

if (theBodyShouldBeEvaluated()) {
return EVAL_BODY_INCLUDE;
} else {
return SKIP_BODY;
}
}

/** .
*
*
* This method is called after the JSP engine finished processing the tag.
* @return EVAL_PAGE if the JSP engine should continue evaluating the JSP page,
otherwise return SKIP_PAGE.
* This method is automatically generated. Do not modify this method.
* Instead, modify the methods that this method calls.
*
*
*/
public int doEndTag() throws JspException, JspException {
otherDoEndTagOperations();

if (shouldEvaluateRestOfPageAfterEndTag()) {
return EVAL_PAGE;
} else {
return SKIP_PAGE;
}
}

public String getNombreTransaccion() {
return nombreTransaccion;
}

public void setNombreTransaccion(String value) {
nombreTransaccion = value;
}
}
```

No olvidar modificar el fichero **web.xml** para poder utilizar las tags en nuestras páginas JSP

```
.....

  <taglib>
     <taglib-uri>/roberto/tokens</taglib-uri>
     <taglib-location>/WEB-INF/tokens.tld</taglib-location>
   </taglib>

</web-app>
```

Como habréis podido observar el efecto es el deseado aunque todavía quedarían por dar muchos pasos:

- Deberíamos construir un juego de pruebas detallado que nos garantizase su correcto comportamiento.
- Como segundo paso deberíamos sacar del código de las TAGs a librerías más genéricas para utilizarlas en aplicaciones tipo MVC y otro tipo de interfaces.
- Como tercer paso, deberíamos ligar estas capacidades dentro de nuestro FrameWorks para que incluso se generase automáticamente el código redundante en el cliente
- Y aún podríamos hacer más cosas....

Control de transacciones y Struts

Mucha gente utiliza Struts pensando que el control de transacciones y doble clicks ya esta solucionado.... pero no es así de automático.

En la acción que deseamos que active el control transaccional (previo al formulario) debemos llamar a la función:

```
saveToken(httpServletRequest);
```

Los formularios tenemos que montarlos con la etiqueta de Struts html

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>

<html>
<head>
<title></title>
</head>

<body>
<center>
<h2>Introduzca los datos</h2>
<hr width='60%'>

<html:form action='/primeraAccion' >
<br>Usuario: <html:text property="usuario"/>
<br>Password: <html:password property="password" redisplay="false"/>
<br><html:submit value="Enviar"/>
</html:form>
<br>
<html:errors/>

</center>
</body>
</html>
```

Podemos ver que pasa algo similar a lo que hicimos con nuestras tags

```
<html>
<head>
<title></title>
</head>

<body>
<center>
<h2>Introduzca los datos</h2>
<hr width='60%'>

<form name="losdatos" method="post" action="/primeraAccion.do">

<input type="hidden"

name="org.apache.struts.taglib.html.TOKEN"

value="2d5b61e821accd2a449299ee78055b8f">
<br>Usuario: <input type="text" name="usuario" value="1234123">
<br>Password: <input type="password" name="password" value="">
<br><input type="submit" value="Enviar">
</form>
<br>
<UL><LI>Hay un problema con la password</LI></UL>

</center>
</body>
</html>
```

Posteriormente en la acción que ejecuta la transacción podemos verificar el token e invalidarlo con:

```
if( this.isTokenValid(httpServletRequest)==true)
```

Y luego limpiarlo con

```
this.resetToken(httpServletRequest);
```

Conclusiones

El desarrollo de aplicaciones Web es un arte bastante compleja aunque aparentemente pudiera parecer lo contrario Uno de los problemas es que la misma cosa se puede hacer de muchos modos y es fácil que elijamos uno poco afortunado.

Este ejemplo no soluciona todos los problemas pero sienta unas bases para el estudio y solución del mismo..

Si estudiáis metodologías tipo Programación Extrema, observareis que uno de los principios a tener en cuenta es realizar diseños sencillos. Esto significa (y es una posible interpretación personal) que no debemos diseñar una solución pensando en los problemas que tendremos dentro de dos años sino los que tendremos dentro de dos meses... Dentro de dos años ya veremos....

Nuevo servicio de notificaciones

Si deseas que te enviemos un correo electrónico cuando introduzcamos nuevos tutoriales, inserta tu dirección de correo en el siguiente formulario.

Subscribirse a Novedades	
e-mail	
	Enviar

Otros Tutoriales Recomendados (También ver todos)

Nombre Corto	Descripción
JSP 2.0, JSTL y Lenguaje de expresiones	Os mostramos las novedades de JSP 2.0: Nuevas librerías estandar de etiquetas y el lenguaje de expresiones con ejemplos de acceso a base de datos, XML y XSL en JSP
Novedades en Java 1.5	Ya está disponible la versión Beta del J2SDK 1.5. Os mostramos algunas de las nuevas características introducidas en el lenguaje Java: Clases genéricas, enumeraciones, bucles simplificados, etc.
Forzar diálogo Guardar Como en JSPs	Os mostramos como afrontar tareas comunes en JSP: Forzar el diálogo Guardar Como al generar dinámicamente un fichero desde un JSP y asegurarnos que no se cachean nuestros ficheros (probado en IExplorer 6)
Ejecutar JSPs almacenados en Base de Datos	Atendiendo una pregunta de nuestro foro, os mostramos como, con unos sencillos pasos, podemos ejecutar JSPs almacenandos en la base de datos. Esto puede ser una idea base para un gestor de contenidos construido en Java.
Introducción a Hibernate	Cesar Crespo nos enseña como utilizar unos de los sistemas más extendidos de mapeo de objetos a estructuras relacionales (tablas de base de datos)
Plantear una aplicación Web y Struts	Os mostramos un posible modo de plantear una aplicación Web (análisis) y darla forma. El FrameWork utilizado es struts y tratamos de identificar qué depende de este FrameWork y qué no.
Despliegue gráfico de EJBs	Os mostramos como crear y desplegar de un modo gráfico un EJB de sesión el el servidor de aplicaciones de referencia de Sun
JDO con OJB	Os mostramos como configurar el entorno OJB de apache para construir la primera aplicación JDO
Gestión de contenidos y errores comunes	Os explicamos en que consiste la gestión de contenidos y cuales son los errores cometidos por multitud de empresas a la hora de abordar su implantación
Patrones de diseño J2EE	Os mostramos una interpretación particular de los patrones de diseño J2EE

www.AdictosAlTrabajo.com Opimizado 800X600