Ejemplo de Filtros

Dentro de la especificación de Java Servlets, existe un componente llamado filtros. Estos nos ayudan a analizar o transformar las peticiones hechas a las páginas Web, ya sean jsps o servlets. Estos filtros trabajan de forma encadenada, por lo que se puede crear una serie de filtros que van pasando el control de uno al otro.

Los Filtros son muy sencillos de implementar, y son componentes altamente reutilizables que nos dan mucho poder sobre las aplicaciones web. Nos permite por ejemplo, controlar el acceso a la aplicación, manejo de compresión de datos, transformaciones, auditoria, registro, cifrado, manejo de sesiones, en fin, un sinnúmero de utilidades que se adaptan a nuestras necesidades.

En este ejemplo veremos como implementar un filtro que nos ayudara con una sencilla tarea: detectar el timeout de nuestra sesión. Cuando un usuario deja morir su sesión, no teniendo actividad en 30 minutos por ejemplo, esta se destruye, causando que si el usuario solicita una página que maneje datos de la sesión, esta regresara errores y resultados no esperados.

Empecemos creando nuestro filtro. Lo llamaremos SessionFilter.

Tenemos que crear una clase que implemente la interfaz javax.servlet.Filter. Esta interfaz contiene 3 métodos:

doFilter: Este es el mas importante ya que es el que realiza toda la actividad del filtro. Además de disparar el resto de los filtros.

Init: Este método es llamado cuando el filtro es inicializado en el contenedor.

Destroy: Este método es llamado cuando el filtro es destruido por el contenedor.

Nuestro filtro vacio quedaria como sigue, nota el chain.doFilter, este ejecuta el resto de los filtros pendientes:

package net.novacreations.example.filter;

import javax.servlet.Filter;  
import javax.servlet.FilterChain;  
import javax.servlet.FilterConfig;

import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;

public class SessionFilter implements Filter {

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {  
try {  
//Podemos seguir ejecutando los demas filtros si asi lo necesitamos  
chain.doFilter(request, response);  
} catch (IOException e) {  
e.printStackTrace();  
} catch (ServletException e) {  
e.printStackTrace();  
}

}

public void destroy() {  
}

public void init(FilterConfig arg0) throws ServletException {  
}

}

Una vez creado nuestro filtro, tenemos que definirlo en el web.xml, así que vamos a este archivo. Además de la definición de nuestros servlets y otros atributos, debemos de indicarle los filtros, y sus mapeos:  
  
  
nombreFiltro  
claseDelFiltro

nombreDelFiltro  
/patronDeLaUrl

Para nuestro ejemplo, quedaría así:  
  
  
sessionFilter  
net.novacreations.example.filter.SessionFilter

sessionFilterVariable  
/sessionTest

Ya con esto, nuestro filtro será inicializado la próxima vez que hagamos un deploy. Este se ejecutara cada vez que una solicitud concuerde con la url-pattern, en este caso siempre que se ejecute el servlet sessionTest. Pero también podríamos pedirle que se ejecute con todos los jsps, htmls o todos los servlets, con el patrón que necesitemos.

Ahora ya estamos listos para meterle código a nuestro filtro. En este caso nuestro servlet sessionTest asume que la sesión existe, por lo que tenemos que verificar en cada solicitud si esto es cierto y en caso contrario mandarlo a una pagina con un aviso de timeout.

Podemos mandar a pedir la sesión, y si esta no existe, podemos considerar que ha sido eliminada.

HttpServletRequest httpRequest = (HttpServletRequest)request;

HttpSession session= httpRequest.getSession(false);

Al mandarle false como parámetro, evitamos que cree una nueva sesión en blanco. Así que si sesión es null, nuestra sesión ha expirado.

Tenemos que considerar que si tenemos jsps, la lógica puede cambiar. Los jsps al ser cargados crean por defecto una sesión si es que esta no existe. Por lo que si nuestra aplicación carga jsps antes de llegar a nuestro servlet, va a crear una nueva sesión sin relación con la sesión expirada, por lo que los datos que necesitamos ya se perdieron. Para esto podemos preguntar además por una variable que nos sirve de bandera, por ejemplo el nombre de usuario o algún otro dato que nuestra aplicación siempre maneje. En este caso, si ese dato no existe, podemos considerar la sesión como perdida.

Object usuario = session.getAttribute(“nombre”);

Aquí si el usuario es nulo, nuestra sesión esta perdia.

Por lo que nuestro doFilter quedaría así:  
  
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {

System.out.println("FILTER");

//Cargamos la sesion  
HttpServletRequest httpRequest = (HttpServletRequest)request;  
HttpSession session= httpRequest.getSession(false);

Object usuario = null;  
//Preguntamos por un objeto en la sesion  
if(session!=null)usuario=session.getAttribute("nombre");

if(session==null || usuario==null)System.out.println("Session Invalid");  
else System.out.println("Session valid");

if (session==null || usuario==null) {//if we have no session

try {  
httpRequest.getRequestDispatcher("/timeout.html").forward(request, response);  
return;  
}  
catch (ServletException e) { }  
catch (IOException e) {}  
}

try {  
chain.doFilter(request, response);  
} catch (IOException e) {  
e.printStackTrace();  
} catch (ServletException e) {  
e.printStackTrace();  
}  
}

Con el método httpRequest.getRequestDispatcher lo que hacemos es mandar como respuesta una url diferente, en este ejemplo en mensaje de timeout.

Una vez cargado este filtro dentro de nuestra aplicación, este mostrara el mensaje de Timeout cada vez que el usuario deje morir su sesión al no tener actividad sobre la misma. Puedes ver un ejemplo listo para el deploy en  
<http://www.novacreations.net/192-ejemplo-de-filtros-en-aplicacion-web-java-servlets/> Cuenta con todas los códigos fuentes.

Para generar un Filtro es necesario que la Clase en cuestión herede ("inherit") el comportamiento de la clase llamada Filter, através de dicha Clase se tiene acceso a los métodos y mecanismos ofrecidos por un filtro. Este procedimiento es muy similar al de un Servlet que hereda ("inherit") su comportamiento de la Clase *HttpServlet* ( o *GenericServlet*).Los tres métodos que debe implementar un filtro son:

* **doFilter** (*Obligatorio*): Este método es la parte medular de todo Filtro ya que dentro de él se incluyen las tareas principales de ejecución.
* **init** (*Opcional*) : Es un método ejecutado antes del método *doFilter*, su labor principal es adquirir/inicializar algún recurso que será empleado por *doFilter*.
* **destroy**(*Opcional*) : Ejecutado una vez que ha terminado el método *doFilter*, su labor es liberar los recursos utilizados/adquiridos en el proceso de ejecución los cuales generalmente son aquellos reservados por *init*.

**Código Fuente ImpuestosRegistro.java.**

|  |
| --- |
| package com.osmosislatina.filtros;  import java.io.\*;  import javax.servlet.\*;  import javax.servlet.http.\*;  import com.osmosislatina.beans.\*;  import java.util.\*;  public class ImpuestosRegistro implements Filter  {  private FilterConfig config;    public void init(FilterConfig config) throws ServletException {  this.config = config;  }    public void destroy() {  config = null;  }    public void doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) throws IOException, ServletException {  chain.doFilter(request, response);  // Definir variables para colocar nota de impuestos  int impuestos = 0;  String nota = "";  // Extraer si filtro incluye impuestos  try {  impuestos = Integer.parseInt(config.getInitParameter("impuestos"));  } catch (NumberFormatException exc) {  config.getServletContext().log("No fue posible convertir parámetro de filtro a numero entero", exc);  }      if (impuestos > 0) {  // Existe una tarifa de impuestos definida en filtro  nota = " Aplica " + impuestos + " % impuestos ";  }    // Extraer Sesión  HttpSession session = ((HttpServletRequest)request).getSession();  session.setAttribute("impuesto",nota);      // Datos de la Puja  Date fecha = new Date();  String nombre = request.getParameter("producto");  String cantidad = request.getParameter("cantidad");  String precio = request.getParameter("precio");    String puja = " \* Puja Realizada : " + fecha + " / Producto : " + nombre +  " / Cantidad : " + cantidad + " / Precio Unitario : $" + precio;        config.getServletContext().log("Registro PUJA: " + puja);  }  } |

* Primeramente se define la librería ("package") a la cual pertenecerá esta Clase, en este caso com.osmosislatina.filtros, para posteriormente declarar la Clase principal ImpuestoRegistro que implementa la Interfase javax.servlet.Filter una característica primordial de todo filtro.
* Es definido un campo del tipo FilterConfig que contendrá datos de configuración acerca del filtro.
* Se declara el método init que toma como dato de entrada un objeto de tipo FilterConfig el cual es asignado al campo definido anteriormente. Vale mencionar que dicho dato de entrada proviene de los datos de configuración del filtro según definidos en *web.xml*.
* Posteriormente se define el método destroy que asigna un valor nulo (null) al campo config.
* El método principal doFilter toma como datos de entrada un objeto ServletRequest que representa la requisición, un objeto ServletResponse que contiene el objeto de respuesta, y otro objeto FilterChain que representa una cadena de filtros; la lógica definida en este método se describe a continuación :
  + Como primer paso se invoca el mismo métdo doFilter sobre la cadena de filtros (FilterChain), el cual toma como parámetros la requisición y respuesta proporcionados como datos de entrada. Esta invocación simplemente indica que debe ser aplicado el filtro. NOTA: El grupo de filtros o cadena a la que se refiere FilterChain es definida en el archivo *web.xml*
  + Se definen dos variables para colocar configuraciones y valores del filtro, seguido se declara un bloque *try/catch* para extraer el parémetro impuestos definido en la configuración del filtro (definido en *web.xml*). Nótese que dicho parámetro es extraído através del método getInitParameter del objeto config.
  + En base al valor de configuración se asigna este valor a la variable nota, para posteriormente ser colocado en la sesión del usuario bajo el nombre de impuesto. NOTA: Observe que la sesión del usuario, aunque extraída del objeto request, se realiza através de un *cast* hacia HttpServletRequest, lo anterior es necesario ya que el objeto request de un filtro proviene de la interfase genérica ServletRequest más no de la sub-interfase HttpServletRequest.
  + Posteriormente se genera un objeto con la fecha del sistema, y a su vez se extraen de la requisición (request) los parámetros producto, cantidad y precio, los cuales son colocados en un String llamado puja.
  + Finalmente, mediante el objeto config se accesa el registro ("Log") del "Servlet Engine" através del método getServletConfig para enviar el valor antes generado (puja) hacia el registro.

En la siguiente sección se describe como es configurado este filtro en el archivo de configuración *web.xml*.

# Desarrollo de filtros de servlets para el proceso de inicio de sesión de formulario

Utilizando el mecanismo de inicio de sesión basado en formulario, puede controlar el diseño de la pantalla de inicio de sesión. En el inicio de sesión basado en formulario, debe especificar una página de inicio de sesión que se utiliza para recuperar la información sobre el ID de usuario y la contraseña. También se puede especificar una página de error que se visualice cuando la autenticación no sea correcta.

## Acerca de esta tarea

Si se requiere una autenticación adicional o un proceso adicional antes y después de la autenticación, se pueden utilizar los filtros de servlets. Los filtros de servlets pueden utilizarse para interceptar de forma dinámica las solicitudes y las respuestas para transformar o utilizar la información contenida en las solicitudes o las respuestas. Uno o más filtros de servlets pueden adjuntarse a un servlet o a un grupo de servlets. Los filtros de servlets también se pueden adjuntar a archivos JSP y páginas HTML. Antes de invocar al servlet se llama a todos los filtros de servlets que se han adjuntado.

Tanto el filtro de servlet como el filtro de inicio de sesión basado en formulario reciben soporte en todos los contenedores web que cumplan con la especificación de servlet versión 2.3. El servlet de inicio de sesión basado en formulario lleva a cabo la autenticación y los filtros de servlets pueden efectuar tareas adicionales de autenticación, auditoría o de registro cronológico de la información.

Para llevar a cabo acciones previas y posteriores al inicio de sesión con los filtros de servlet, configure estos filtros para que den soporte a la página de inicio de sesión basado en formulario o para el URL /j\_security\_check. El URL j\_security\_check se envía mediante una página de inicio de sesión basado en formulario con el parámetro j\_username que contiene el nombre de usuario y el parámetro j\_password que incluye la contraseña. Un filtro de servlet puede utilizar el parámetro de nombre de usuario y la información de contraseña para realizar otras tareas de autenticación o para otras necesidades especiales.

## Procedimiento

1. Un filtro de servlet implementa la clase javax.servlet.Filter. Hay tres métodos en la clase de filtro que es necesario implementar:
   * **init(javax.servlet.FilterConfig cfg)**. Este método lo llama el contenedor una vez cuando el filtro de servlet entra en funcionamiento. El elemento FilterConfig que se pasa a este método contiene los init-parameters del filtro de servlet. Los init-parameters se pueden especificar para un filtro de servlets durante la configuración mediante la herramienta de ensamblaje.
   * **destroy**. Este método lo llama el contenedor cuando el filtro de servlets deja de funcionar.
   * **doFilter(ServletRequest req, ServletResponse res, FilterChain chain)**. El contenedor llama a este método para cada solicitud de servlet correlacionada con este filtro antes de invocar el servlet. La cadena FilterChain que se pasa a este método puede utilizarse para invocar el filtro siguiente en la cadena de filtros. El servlet original solicitado se ejecuta cuando el último filtro de la cadena llama al método chain.doFilter. Por lo tanto, todos los filtros deben llamar al método chain.doFilter para poder ejecutar el servlet original después del filtrado. Si se implementa una comprobación de autenticación adicional en el código de filtro y se produce una anomalía, no es necesario ejecutar el servlet original. No se llama al método chain.doFilter y éste se puede redireccionar a alguna página de error diferente.
2. Si un servlet se correlaciona con muchos filtros de servlets, se llama a los filtros de servlets en el orden en que se enumeran en el descriptor de despliegue web.xml de la aplicación. El archivo de clases de filtros de servlets debe colocarse en el directorio WEB-INF/classes de la aplicación.

## Ejemplo

**Un ejemplo de un filtro de servlet.**

Este filtro de inicio de sesión puede correlacionarse con el URL /j\_security\_check para realizar acciones previas y posteriores al inicio de sesión.

**import** javax.servlet.\*;

**public** **class** LoginFilter **implements** Filter {

**protected** FilterConfig filterConfig;

// Se llama cuando se crea una instancia del filtro.

// Si se correlaciona con j\_security\_check, se llama

// la primera vez que se invoca j\_security\_check.

**public** **void** init(FilterConfig filterConfig) **throws** ServletException {

**this**.filterConfig = filterConfig;

}

**public** **void** destroy() {

**this**.filterConfig = **null**;

}

// Se llama por cada solicitud correlacionada a este filtro.

// Si se correlaciona con j\_security\_check,

// se llama para cada acción j\_security\_check

**public** **void** doFilter(ServletRequest request,

ServletResponse response, FilterChain chain)

**throws** java.io.IOException, ServletException {

// efectuar aquí la acción previa al inicio de sesión

chain.doFilter(request, response);

// llama al siguiente filtro de la cadena.

// j\_security\_check si este filtro

// se correlaciona con j\_security\_check.

// efectuar aquí la acción posterior al inicio de sesión.

}

}

Copy

**Utilización de filtros de servlet para realizar procesos anteriores y posteriores al inicio de sesión durante el inicio de sesión de formularios**

Este ejemplo ilustra un modo en que los filtros de servlet pueden realizar los procesos anteriores y posteriores al inicio de sesión durante el inicio de sesión de formularios.

Servlet filter source code: LoginFilter.java

/\*\*

\* Ejemplo de filtro de servlet: Este ejemplo filtra j\_security\_check y

\* realiza acciones anteriores al inicio de sesión para determinar si el usuario que está intentando iniciar la sesión

\* está en la lista de revocados. Si el usuario está en la lista de revocados, se envía un error

\* al navegador.

\*

\* Este filtro lee el nombre de archivo de la lista de revocados en la FilterConfig

\* que se pasa en el método init(). Lee el archivo de la lista de usuarios revocados y

\* crea una lista revokedUsers.

\*

\* Cuando se llama al método doFilter, se comprueba el usuario que está iniciando la sesión

\* para confirmar que éste usuario no está en la lista de usuarios revocados.

\*

\*/

**import** javax.servlet.\*;

**import** javax.servlet.http.\*;

**import** java.io.\*;

**public** **class** LoginFilter **implements** Filter {

**protected** FilterConfig filterConfig;

java.util.List revokeList;

/\*\*

\* init() : se llama al método init() cuando se crea una instancia del filtro.

\* Se crea una instancia de este filtro la primera vez que se invoca j\_security\_check para

\* la aplicación (es decir, cuando se accede a un servlet protegido

\* de la aplicación).

\*/

**public** **void** init(FilterConfig filterConfig) **throws** ServletException {

**this**.filterConfig = filterConfig;

// leer la lista de usuario revocados

revokeList = **new** java.util.ArrayList();

readConfig();

}

/\*\*

\* destroy() : se llama al método destroy() cuando el filtro

\* ya no está en servicio.

\*/

**public** **void** destroy() {

**this**.filterConfig = **null**;

revokeList = **null**;

}

/\*\*

\* doFilter() : se llama al método doFilter() antes de que se invoque el servlet

\* con el que está correlacionado este filtro. Dado que este filtro se

\* correlaciona con j\_security\_check, se llama a este método antes de

\* de que se envíe la acción j\_security\_check.

\*/

**public** **void** doFilter(ServletRequest request, ServletResponse response,

FilterChain chain) **throws** java.io.IOException, ServletException {

HttpServletRequest req = (HttpServletRequest)request;

HttpServletResponse res = (HttpServletResponse)response;

// acción de inicio de sesión previa

// obtener nombre de usuario

String username = req.getParameter("j\_username");

// si el usuario está en la lista de revocados, enviar error

**if** ( revokeList.contains(username) ) {

res.sendError(javax.servlet.http.HttpServletResponse.SC\_UNAUTHORIZED);

**return**;

}

// llamar al siguiente filtro de la cadena: permitir que j\_security\_check autentique al

// usuario

chain.doFilter(request, response);

// acción de inicio de sesión posterior

}

/\*\*

\* readConfig() : Lee el archivo de la lista de usuarios y crea una lista

\* de usuarios revocada.

\*/

**private** **void** readConfig() {

**if** ( filterConfig != **null** ) {

// obtener el archivo de la lista de usuarios revocados y abrirlo.

BufferedReader **in**;

**try** {

String filename = filterConfig.getInitParameter("RevokedUsers");

**in** = **new** BufferedReader( **new** FileReader(filename));

} **catch** ( FileNotFoundException fnfe) {

**return**;

}

// leer todos los usuarios revocados y añadirlos a revokeList.

String userName;

**try** {

**while** ( (userName = **in**.readLine()) != **null** )

revokeList.add(userName);

} **catch** (IOException ioe) {

}

}

}

}

Copy

**Importante**

En el ejemplo de código anterior, la línea que empieza por public void doFilter(ServletRequest request se separa en dos sólo a efectos ilustrativos. La línea public void doFilter(ServletRequest request y la línea siguiente son una sola línea.

Ejemplo del archivo web.xml que muestra el filtro LoginFilter configurado y correlacionado con el URL j\_security\_check:

<filter id="Filter\_1">

<filter-name>*LoginFilter*</filter-name>

<filter-class>LoginFilter</filter-class>

<description>Realiza una operación previa y posterior al inicio de sesión</description>

<init-param>

<param-name>RevokedUsers</param-name>

<param-value>c:\WebSphere\AppServer\installedApps\

<app-name>\revokedUsers.lst</param-value>

</init-param>

</filter-id>

<filter-mapping>

<filter-name>LoginFilter</filter-name>

<url-pattern>/j\_security\_check</url-pattern>

</filter-mapping>