Publicado en BYTE

Filtros y Servlets en Java

Por Oscar González Moreno (oscar.gonzalez@samelan.com).
URL: http://www.samelan.com/oscargonzalez

El desarrollo de aplicaciones Web con Java, pasa en algún momento por el uso del sistema de Servlets. Esta especificación ha ido depurándose y creciendo según las necesidades y evolución del mercado. Una de las nuevas características introducidas en la versión 2.3 es el concepto de "Filtros". Veamos de qué tratan y cómo utilizarlos en nuestras aplicaciones.

A menudo, el mero uso de los Servlets de Java (o sus páginas JSP) no es suficiente para los requisitos de nuestras aplicaciones. En muchas ocasiones nos encontramos con acciones repetitivas, que hay que realizarlas en todas nuestras páginas. Algunos ejemplos de estas acciones serían un control de seguridad, una herramienta de LOG de operaciones, o un sistema de compresión de datos. La solución tradicional a estos problemas ha sido el uso de ficheros incluidos ("include") que permitían utilizar una misma funcionalidad desde un gran número de páginas JSP o Servlets.

Pero esta solución no deja de resultar poco elegante. Lo óptimo sería contar con un mecanismo, gestionado por el contenedor de Servlets (Tomcat, Resin, etc.) que "interceptara" las peticiones a Servlets o páginas JSP, y que actuara antes y después de pasar el control al Servlet. Esto es precisamente lo que hacen los Filtros ("Filters") de Java.

Una de las mayores ventajas de utilizar el sistema de Filtros, es que su uso resulta totalmente transparente para el desarrollador de los Servlets o páginas JSP. Es decir, imaginemos que tenemos programado un Filtro que encripta los datos (la salida) que generan los Serlvets). En tal caso, el desarrollador del Servlet como tal no tiene que preocuparse de nada mas que la lógica de su propio Servlet, ya que la ejecución y funcionamiento del Filtro es algo totalmente ajeno a él, y un proceso que ocurrirá después de haberse ejecutado su propio Servlet. Así se consigue un buen nivel de transparencia y reutilización de código. Esto nos lleva normalmente a un escenario en el cual, un alto número de Servlets utilizan o comparten uno o varios Filtros.

Los filtros por tanto, son unas clases de Java, que implementan y heredan de unos interfaces y clases específicos que veremos a continuación, y que tienen el siguiente funcionamiento:
Cuando una petición de Servlet llega al servidor, dependiendo de cómo hayamos configurado nuestra aplicación, se ejecutará uno o varios filtros. Cada uno de estos filtros nos permitirá realizar lo siguiente: Revisar y modificar el objeto Request (variables, cabeceras, etc.), revisar y modificar el objeto Response (el flujo de respuesta enviado al cliente), lanzar excepciones, o continuar con la cadena de filtros.
Por tanto, los filtros actúan por un lado como pre-procesadores de los Servlets, o bien como post-procesadores de los mismos (aunque un solo Filtro puede realizar las dos cosas, lo normal es que solo se emplee para una de ellas).

En el sitio web de SUN, podremos encontrar numerosa información sobre los filtros de Java.

Como nota adicional, es preciso destacar que los Filtros de Java también puede procesar contenido estático (páginas HTML, ficheros de imágenes, etc.), lo cual ofrece nuevas y amplias posibilidades, ya que nos permite incorporar desarrollo (programación) a contenidos estáticos.
A continuación vamos a ver un sencillo ejemplo de cómo utilizar los Filtros en una aplicación "real" de servidor en Java. Para ello, vamos a suponer que tenemos un portal de Internet, al que queremos añadir una funcionalidad de LOG de acceso al mismo.
Para ello, necesitamos un código que almacene el número total de accesos a nuestro portal (en nuestro ejemplo, un único Servlet). Para ello vamos a utilizar la funcionalidad de los Filtros de Java. El código de ejemplo es el del listado 1:

```
public final class VisitasFilter implements Filter {

   private FilterConfig filterConfig = null;

   public void init(FilterConfig filterConfig)
      throws ServletException {
      this.filterConfig = filterConfig;
   }

   public void destroy() {
      this.filterConfig = null;
   }

   public void doFilter(ServletRequest request,
      ServletResponse response, FilterChain chain)
      throws IOException, ServletException {

      if (filterConfig == null) {
         return;
      }

      StringWriter sw = new StringWriter();
      PrintWriter writer = new PrintWriter(sw);
      Counter                          counter                          =
(Counter)filterConfig.getServletContext().getAttribute("Visitas");

      writer.println();
      writer.println("===============");
```

```
      writer.println("El número de visitas es: " +
counter.incCounter());
      writer.println("===============");

      writer.flush();
      filterConfig.getServletContext().log(sw.getBuffer().toString());
      chain.doFilter(request, wrapper);
   }
}
```
Listado 1. Filtro de contador de visitas

Como vemos, la programación resulta muy sencilla. En primer lugar, construimos una clase Java que implemente la interfaz "Filter". A continuación, implementamos dos métodos que nos impone dicha interfaz, que son los métodos "init" y "destroy". Dichos métodos serán invocados por el contenedor de Servlets (en nuestro ejemplo estamos utilizando Tomcat) y se ocupan respectivamente de ejecutar las operaciones de inicialización y destrucción de nuestro Filtro.

Pero el método mas importante es el tercer método de la interfaz "Filter", que se denomina "doFilter". Este método recibe tres parámetros, que son el objeto Request de la petición HTTP del usuario, el objeto Response asociado a la misma petición y el objeto FilterChain que representa la cadena de Filtros que están encolados para su ejecución (el sistema de Filtros de Java permite ejecutar varios filtros para un mismo Servlet o página JSP).

Con estros tres objetos, tenemos toda la capacidad para manipular a nuestro gusto tanto la entrada de parámetros, como la salida de contenido, como también el flujo de ejecución de los distintos Filtros asociados a nuestro Servlet. Lo único que queda por hacer es implementar la funcionalidad deseada dentro del citado método "doFilter".

Otro uso muy común que se le da a los Filtros de Java, es el utilizarlos para medir el tiempo de ejecución total de un Servlet o página JSP. En el listado 2 tenemos un ejemplo de un Filtro que realiza precisamente esta operación.

```
public final class MedidaFilter implements Filter
{
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
        throws IOException, ServletException
    {

        long startTime = System.currentTimeMillis();

      chain.doFilter(request, response);

      long stopTime = System.currentTimeMillis();

      System.out.println("Tiempo total de ejecucion: " +
(stopTime - startTime) + " milisegundos");

}
```
Listado 2. Filtro de medición de tiempo de ejecución

Este filtro, lo primero que realiza es ejecutar toda la cadena de ejecución de los Filtros. Esto se hace para que la medición de tiempo de ejecución sea completa, con la totalidad de los filtros implicados en el proceso.

A continuación, una vez que ha terminado todo el proceso, es cuando se saca por la consola de salida estándar el tiempo total que ha tardado en ejecutarse completamente el código.

Configuración

Vista la mecánica básica de los Filtros de Java, falta ver un apartado importante, que es el de la configuración. ¿Cómo le decimos al servidor que queremos utilizar un filtro en concreto para un Servlet?.
Para ello, tenemos que modificar el archivo WEB-INF/web.xml (viejo conocido de los desarrolladores de Servlets). En él, incluiremos un código como el del listado 3:

```
<filter>
<filter-name>MedidaFilter</filter-name>
<filter-class>com.samelan.drdobbs.MedidaFilter</filter-class>
<description>
    Filtro de medición de tiempos de ejecución
</description>
</filter>
```

Listado 3. Configuración del archivo Web.xml para utilizar un Filtro.

El conjunto total de elementos de configuración que podemos utilizar es el recogido en el cuadro 1.

filter-name	El nombre que será utilizado para identificar el filtro de forma única en el archivo web.xml.
filter-class	El nombre de clase del filtro, incluyendo el paquete. Este nombre será utilizado por el contenedor de Servlets para cargar la clase del Filtro.
init-param	Parámetros de inicialización que el contenedor de Serlvets le pasará de forma automática al Filtro.
description	Descripción del Filtro (arbitraria).
Icon	Rutas (opcionales) para archivos de imágenes utilizados por herramientas visuales de programación, para identificar el filtro.
display-name	Nombre (opcional) utilizado por herramientas visuales de programación, para identificar el filtro.

Cuadro 1. Configuración del archivo Web.xml

Con estos breves fragmentos de código, podremos comenzar a trabajar de forma inmediata con toda la potencia de los Filtros de Java.