Servlet Filters are Java classes that can be used in Servlet Programming for the following purposes:

To intercept requests from a client before they access a resource at back end.

To manipulate responses from server before they are sent back to the client.

There are are various types of filters suggested by the specifications:

Authentication Filters.

Data compression Filters.

Encryption Filters.

Filters that trigger resource access events.

Image Conversion Filters.

Logging and Auditing Filters.

MIME-TYPE Chain Filters.

Tokenizing Filters .

XSL/T Filters That Transform XML Content.

Filters are deployed in the deployment descriptor file **web.xml**and then map to either servlet names or URL patterns in your application's deployment descriptor.

When the web container starts up your web application, it creates an instance of each filter that you have declared in the deployment descriptor. The filters execute in the order that they are declared in the deployment descriptor.

## Servlet Filter Methods:

A filter is simply a Java class that implements the javax.servlet.Filter interface. The javax.servlet.Filter interface defines three methods:

|  |  |
| --- | --- |
| **S.N.** | **Method & Description** |
| 1 | **public void doFilter (ServletRequest, ServletResponse, FilterChain)**  This method is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain. |
| 2 | **public void init(FilterConfig filterConfig)**  This method is called by the web container to indicate to a filter that it is being placed into service. |
| 3 | **public void destroy()**  This method is called by the web container to indicate to a filter that it is being taken out of service. |

## Servlet Filter Example:

Following is the Servlet Filter Example that would print the clients IP address and current date time. This example would give you basic understanding of Servlet Filter, but you can write more sophisticated filter applications using the same concept:

// Import required java librariesimport java.io.\*;import javax.servlet.\*;import javax.servlet.http.\*;import java.util.\*;

// Implements Filter classpublic class LogFilter implements Filter {

public void init(FilterConfig config)

throws ServletException{

// Get init parameter

String testParam = config.getInitParameter("test-param");

//Print the init parameter

System.out.println("Test Param: " + testParam);

}

public void doFilter(ServletRequest request,

ServletResponse response,

FilterChain chain)

throws java.io.IOException, ServletException {

// Get the IP address of client machine.

String ipAddress = request.getRemoteAddr();

// Log the IP address and current timestamp.

System.out.println("IP "+ ipAddress + ", Time "

+ new Date().toString());

// Pass request back down the filter chain

chain.doFilter(request,response);

}

public void destroy( ){

/\* Called before the Filter instance is removed

from service by the web container\*/

}}

Compile **LogFilter.java** in usual way and put your class file in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes.

## Servlet Filter Mapping in Web.xml:

Filters are defined and then mapped to a URL or Servlet, in much the same way as Servlet is defined and then mapped to a URL pattern. Create the following entry for filter tag in the deployment descriptor file **web.xml**

<filter>

<filter-name>LogFilter</filter-name>

<filter-class>LogFilter</filter-class>

<init-param>

<param-name>test-param</param-name>

<param-value>Initialization Paramter</param-value>

</init-param></filter><filter-mapping>

<filter-name>LogFilter</filter-name>

<url-pattern>/\*</url-pattern></filter-mapping>

The above filter would apply to all the servlets because we specified **/\*** in our configuration. You can specify a particular servlet path if you want to apply filter on few servlets only.

Now try to call any servlet in usual way and you would see generated log in your web server log. You can use Log4J logger to log above log in a separate file.

## Using Multiple Filters:

Your web application may define several different filters with a specific purpose. Consider, you define two filters *AuthenFilter* and *LogFilter*. Rest of the process would remain as explained above except you need to create a different mapping as mentioned below:

<filter>

<filter-name>LogFilter</filter-name>

<filter-class>LogFilter</filter-class>

<init-param>

<param-name>test-param</param-name>

<param-value>Initialization Paramter</param-value>

</init-param></filter>

<filter>

<filter-name>AuthenFilter</filter-name>

<filter-class>AuthenFilter</filter-class>

<init-param>

<param-name>test-param</param-name>

<param-value>Initialization Paramter</param-value>

</init-param></filter>

<filter-mapping>

<filter-name>LogFilter</filter-name>

<url-pattern>/\*</url-pattern></filter-mapping>

<filter-mapping>

<filter-name>AuthenFilter</filter-name>

<url-pattern>/\*</url-pattern></filter-mapping>

## Filters Application Order:

The order of filter-mapping elements in web.xml determines the order in which the web container applies the filter to the servlet. To reverse the order of the filter, you just need to reverse the filter-mapping elements in the web.xml file.

For example, above example would apply LogFilter first and then it would apply AuthenFilter to any servlet but the following example would reverse the order:

<filter-mapping>

<filter-name>AuthenFilter</filter-name>

<url-pattern>/\*</url-pattern></filter-mapping>

<filter-mapping>

<filter-name>LogFilter</filter-name>

<url-pattern>/\*</url-pattern></filter-mapping>

Java WEB项目制作过程中，可以监听 Web应用事件，能最大程度地控制你的Web应用。    
两个比较重要的 WEB应用事件：

· 应用的启动和停止

· Session的创建和失效  
应用启动事件发生在你的应用第一次被servlet容器装载和启动的时候；停止事件发生在Web应用停止的时候。   
 Session创建事件发生在每次一个新的session创建的时候，类似地Session失效事件发生在每次一个Session失效的时候。为了使用这些Web应用事件为你做些有用的事情，我们必须创建和使用一些特殊的“监听”类。

##### · 监听类，它们是实现了下边两个接口中任何一个接口的简单的java类： javax.servlet.ServletContextListener javax.servlet.http.HttpSessionListener       如果你想让你的类监听应用的启动和停止事件，你就得实现ServletContextListener接口;如果你想让你的类去监听Session的创建和失效事件，那你就得实现HttpSessionListener接口。 让我们看看在这些接口中你必须要实现的方法。 ****1.ServletContextListener :**** 接口包括如下两个方法：

· · public void contextInitialized(ServletContextEvent sce);

· public void contextDestroyed(ServletContextEvent sce);   
   
   如果你实现了一个接口，那你就必须实现它所有的方法。因此，如果你想利用应用的启动和停止事件，你就需要创建一个Java类并实现ServletContextListener接口。下边是这样的一个类的例子：  
   
/\*File : ApplicationWatch.java\*/  
import javax.servlet.ServletContextListener;  
import javax.servlet.ServletContextEvent;  
public class ApplicationWatch implements ServletContextListener {  
public static long applicationInitialized = 0L;  
/\* 应用启动事件 \*/  
public void contextInitialized(ServletContextEvent ce) {  
applicationInitialized = System.currentTimeMillis();  
}  
/\*应用停止事件 \*/  
public void contextDestroyed(ServletContextEvent ce) {}  
}  
  
   在上边的代码中，ApplicationWatch类实现了ServletContextListener接口。它实现了接口中的两个方法，但只用了其中的一个方法，另一个方法中没有写任何代码。这个类把应用启动的时间记录在一个可以从其它应用类中存取应用启动时间的public static变量中。   
  
   我将很快解释如何告诉服务器我们有这个监听类，但首先让我们看看HttpSessionListener接口有什么不同的方法。  
**2.HttpSessionListener :**  
这个接口也只包含两个方法，分别对应于Session的创建和失效：

· public void sessionCreated(HttpSessionEvent se);

· public void sessionDestroyed(HttpSessionEvent se);  
  
   如上边的ApplicationWatch例子那样，我们也创建了一个实现HttpSessionListener接口的类。如下：  
/\*File : SessionCounter.java\*/  
import javax.servlet.http.HttpSessionListener;  
import javax.servlet.http.HttpSessionEvent;  
public class SessionCounter implements HttpSessionListener {  
private static int activeSessions =0;  
/\* Session创建事件 \*/  
public void sessionCreated(HttpSessionEvent se) {  
       activeSessions++;  
}  
/\* Session失效事件 \*/  
public void sessionDestroyed(HttpSessionEvent se) {  
if(activeSessions>0)activeSessions--;  
}  
  
public static int getActiveSessions() {  
return activeSessions;  
}  
}  
  
   在上边的代码中，SessionCounter类实现了HttpSessionListener接口，其目的是计算活动会话的数量。  
   好了，我们已经学习了什么是Web应用事件，有什么接口可以用以及看到了一些实现这些接口的例子。让我们看看如何告诉应用服务器我们有这些监听类。

##### Web.xml :

·   
   我们通过把类路径加入/WEB-INF/web.xml文件的标签<listener>中来告诉服务器我们的监听类。下边是一个web.xml文件的例子：  
<!-- Web.xml -->  
<?xml version="1.0" encoding="ISO-8859-1"?>  
<!DOCTYPE web-appPUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN""http://java.sun.com/j2ee/dtds/web-app\_2.3.dtd">  
<web-app>  
<!-- Listeners -->  
<listener>  
<listener-class>  
com.stardeveloper.web.listener.SessionCounter  
</listener-class>  
</listener>  
<listener>  
<listener-class>  
com.stardeveloper.web.listener.ApplicationWatch</listener-class>  
</listener>  
</web-app>  
   如上所示，在web.xml文件中声明监听类是非常简单的。现在，每次的服务器的启动和停止，会话的创建和失效，配置好的监听类的相应的方法就会被调用。就这么简单！