链滴

过滤器监听器核心知识点汇总

作者：Gao-Eason

原文链接：https://ld246.com/article/1649678847852

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20180718.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h3 id="给初学者的建议">给初学者的建议</h3>

过滤器 Filter 及监听器 Listener 在 JavaWeb 开发中很少直接使用，但经常使用在 java
框架中仿照两者的设计思想而创造出来的其他技术，比如 Struts2 中的拦截器，另外在其他框架中也
到了两者并且充当十分重要的角色，比如在 Struts2 框架中采用过滤器作为核心来加载配置文件，拦
请求资源等等。在 Spring 中利用监听器来加载 Spring 核心配置文件。因此初学者还是很有必要对
两个技术核心内容熟悉并掌握的。

<h2 id="监听器-Listener">监听器 Listener</h2>
<h3 id="概念">概念</h3>
所谓的监听器是指对整个 WEB 环境的监听，当被监视的对象发生改变时，立即调用
应的方法进行处理。
<h3 id="作用">作用</h3>
<blockquote>
每个监听器的具体作用不再一一列举，此处只对监听接口做一个列举汇总，若对于某
监听器有疑惑或者感兴趣，自己可以写个例子做下测试。
</blockquote>

监听域对象创建与销毁

ServletContextListener
HttpSessionListener
ServletRequestListener
需要在 web.xml 中配置对应标签

监听域对象属性变更(添加，替换，删除)

ServletContextAttributeListener
HttpSessionAttributeListener
ServletRequestAttributeListener
需要在 web.xml 中配置对应标签

实现指定接口 JavaBean，从 session 作用域存放或异常监听

HttpSessionBindingListener
该监听器需配置到对应的 JavaBean 上

监听特殊 JavaBean 在 session 作用域的活化与钝化

HttpSessionActivationListener
该监听器需配置到对应的 JavaBean 上

<h2 id="过滤器Filter">过滤器 Filter</h2>
<h3 id="过滤器的配置">过滤器的配置</h3>

定义一个实现 Filter 接口的实现类
实现对应的 <code>init(),doFilter(),destory()</code>** 方法**

在 web.xml 中配置 <code><filter></code>** 标签及对应的**<code><filter-mapping></code>** 标签**

<h3 id="Filter生命周期">Filter 生命周期</h3>
<blockquote>
过滤器从创建到销毁的过程
</blockquote>

服务器启动的时候，服务器就会创建过滤器的对象，每次访问被拦截目标资源，过滤中的 doFilter 的方法就会执行。当服务器关闭的时候，服务器就会销毁 Filter 对象。
服务器在启动时执行初始化方法，init。
访问资源被拦截时执行拦截方法，doFiter。放行：chain.doFilter(request,response)
服务器关闭时执行销毁方法，destory

注意：FilterChain 过滤器链中的过滤器的执行的顺序跟 <<strng>filter-mapping> 的配置顺序有关,顺序在前先执行，顺序在后，后执行

过滤器的配置

url-pattern 的配置与 servlet 中的配置一样：

完全路径匹配：以 <code>/</code> 开始 <code>/aa</code> ,<code>/aaa/bbb</code>
目录匹配：以 <code>/</code>** 开始，<code>/*</code> ,**<code>/aaa/*</code>
扩展名匹配：不能以 <code>/</code>** 开始，<code>*.do<code>,<code>*.jsp</code>,<code>*.action</code> **

<h3 id="有关Filter中对doFilter方法的理解">有关 Filter 中对 doFilter 方法的理解</h3>
<pre><code class="highlight-chroma"> public class DemoFilter implements javax.servlet.Filter {
     public void destroy() {
    }
     public void doFilter(javax.servlet.ServletRequest req, javax.servlet.ServletResponse resp, javax.servlet.FilterChain chain) throws javax.servlet.ServletException, IOException {
         HttpServletRequest httpServletRequest = (HttpServletRequest) req;
         System.out.println(httpServletRequest.getRequestURI()+"走进来了~~~~");
         chain.doFilter(req, resp);
         //程序代码...
    }

    &bsp;public void init(javax.servlet.FilterConfig config) throws javax.servlet.ServletException {

     }
 }
</code></pre>

思考：上述代码中，若是在该 Filter 中在执行 <code>chain.doFilter(req,resp)</code>** 之后再加一下程序代码，那么这些程序代码的执行时机是什么时候呢？**

可以这么理解：该 web 应用中所有的过滤器会形成一条按照在 web.xml 中配置 <filter-mapping> 的前后顺序组成的链，每执行到一个 <code>chain.doFilter(req,resp)</code>** 的时候就会去执行下一个过滤器程序，当下一个滤程序执行完毕后会返回到本程序接着走下面的**<code>程序代码</code>** ，所以说过滤器链是个有去有回的过程。**

