JSP 中的 Filter 过滤器和 Listener 监听器

1. JSP 中的过滤器

1.1 什么是过滤器

过滤器是一个程序，它先于与之相关的 servlet 或 JSP 页面运行在服务器上。过滤器可附加到一个或多个 servlet 或 JSP 页面上，并且可以检查进入这些资源的请求信息。在这之后，过滤器可以作如下的选择：

（1）以常规的方式调用资源（即，调用 servlet 或 JSP 页面）。

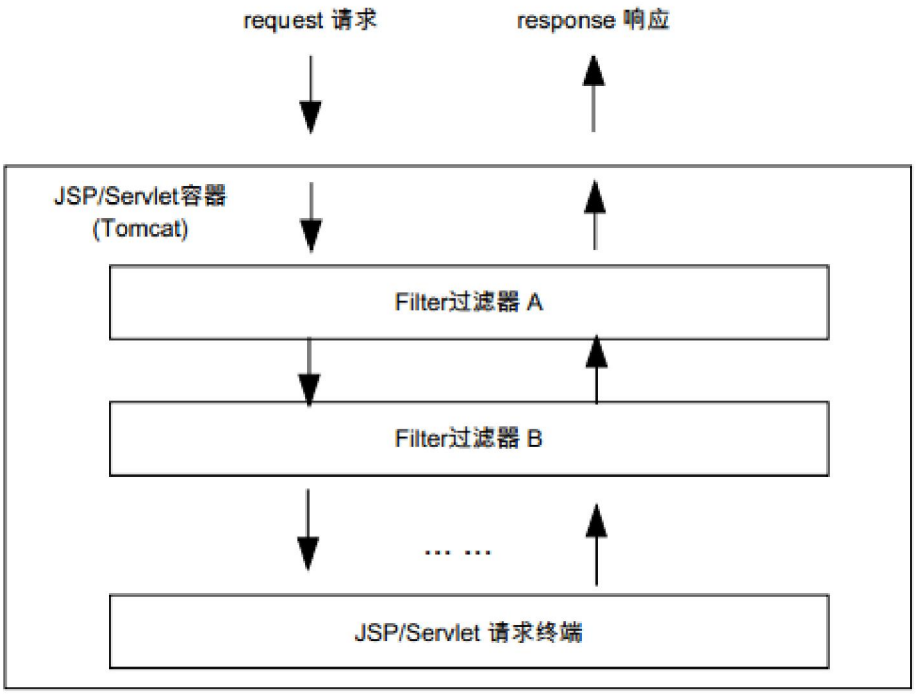
（2）利用修改过的请求信息调用资源。

（3）调用资源，但在发送响应到客户机前对其进行修改。

（4）阻止该资源调用，代之以转到其他的资源，返回一个特定的状态代码或生成替换输出。

1.2 过滤器的基本原理

过滤器可以对客户的请求进行处理，处理完成后，它会交给下一个过滤器处理。这样，客户的请求在“过滤器链”里逐个处理，直到请求发送到目标为止。例如，某网站里有提交“修改业务数据”的网页，当用户填写完修改信息并提交后，服务器在进行处理时需要做两项工作：判断客户端的会话查看该用户是否有修改权限；对提交的数据进行统一编码。这两项工作可以在由两个过滤器组成的过滤链里进行处理。当过滤器处理成功后，把提交的数据发送到最终目标；如果过滤器处理不成功，将把视图派发到指定的错误页面。



1.3 过滤器的使用

开发 Servlet 过滤器的步骤如下：（1）编写实现 Filter 接口的类；（2）在 web.xml 中配置 Filter。

1.3.1 实现 Filter 接口类

Filter 接口定义了以下方法：

成员	描述
destory()	由 Web 容器调用，初始化此 Filter。
init(FilterConfig filterConfig)	由 Web 容器调用，初始化此 Filter。
doFilter(ServletRequest request, ServletResponse response,FilterChain chain)	具体过滤处理代码，其中 FilterChain 参数非常重要，允许通过当前过滤器时须要调用 FilterChain.doFilter()

下面示例实现一个权限过滤器，若用户尚未登录（Session 中没有保存用户信息），将回到登录页面；若已经登录则继续该请求。

```
public class SecurityFilter implements Filter {
```

```
        public void destroy() { }
        public void doFilter(ServletRequest req, ServletResponse resp,
                FilterChain chain) throws IOException, ServletException {
            HttpServletRequest request = (HttpServletRequest) req;
            HttpSession session = request.getSession();
            if(session.getAttribute("user")==null){
                request.getRequestDispatcher("/login.jsp").forward(request, resp);
            }else{
                chain.doFilter(req, resp);
            }
        }
        public void init(FilterConfig arg0) throws ServletException { }
    }
```

1.3.2 在 web.xml 中配置 Filter

要使得 Filter 生效，还必须在 web.xml 中对其进行配置，告知服务器，该过滤器应用在什么模式的 URL 请求上。

```
    <filter>
        <filter-name>securityFilter</filter-name>
        <filter-class>com.securityDemo.filter.SecurityFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>securityFilter</filter-name>
        <url-pattern>/admin/*</url-pattern>
    </filter-mapping>
```

这一段 Filter 的配置和 Servlet 的配置很相似，其中“蓝字”部份声明了须要应用的过滤器类，“红字”部份声明了该过滤器所应用的 URL 路径，就是说在/admin/虚拟目录下的所有请求都要经过该过滤器。

2. JSP 中的监听器

Web 程序在服务器运行的过程中，程序内部会发生多事件，如 Web 应用的启动和停止、Session 会话的开始和销毁、用户请求的开始和结束等等。有时程序员需要在这些事件发生的时机执行一些处理，以完成特定的任务（如通过监控 Session 的开始和结束，可统计网站的在线人数）。事实上，这些事件是可以编码处理的，Servelt API 提供了大量的监听器 Listener 来监听 Web 程序中发生的这些事件，程序员只要实现恰当的特定的事件监听器，就可以对该事件进行处理。

使用监听器需要两个步骤：第一，实现特定的 Listener 类，编写事件处理；第二，通过 web.xml（或者 Annotation）配置启用该 Listener。

2.1. 实现 Listener。

在 JSP 2.0/Servlet 2.4 中，共有八个 Listener 接口，六个 Event 事件类别，实现 Listener 时，需实现对应的接口。这些接口及其功能，详见下表所示：

监听器接口	实现方法	事件	执行时机
ServletContextListener	contextInitialized() contextDestroyed()	ServletContextEvent	加载 Web 应用时（如启动服务器后），会调用 contextInitialized()，移除 Web 应用时（服务器停止），会调用 contextDestroyed ()方法。
ServletContextAttributeListener	attributeAdded() attributeReplaced() attributeRemoved()	ServletContextAttributeEvent	向 application 设置属性、置换、移除属性时依次调用这三个方法
HttpSessionListener	sessionCreated()	HttpSessionEvent	在 HttpSession 对象创建和销

	sessionDestroyed ()		毁时会依次调用这两个方法
HttpSessionAttributeListener	attributeAdded() attributeReplaced() attributeRemoved()	HttpSessionBindingEvent	向 Session 设置属性、置换、移除属性时依次调用这三个方法
HttpSessionActivationListener	sessionDidActivate() sessionWillPassivate()	HttpSessionEvent	当 session 对象为了资源利用或负载平衡等原因而必须暂时储存至硬盘或其它储存器时（透过对象序列化），所作的动作称之为 Passivate，而硬盘或储存器上的 session 对象重新加载 JVM 时所采的动作称之为 Activate，所以，这两个方法分别执行于 Activeate 之后与 Passivate 之前
ServletRequestListener	requestInitialized() requestDestroyed()	RequestEvent	在 HttpServletRequest 对象创建和销毁时会依次调用这两个方法
ServletRequestAttributeListener	attributeAdded() attributeReplaced() attributeRemoved()	ServletRequestAttributeEvent	向 request 对象设置属性、置换、移除属性时依次调用这三个方法
HttpSessionBindingListener	valueBound() valueUnbound()	HttpSessionBindingEvent	其实例被加入至 session 对象的属性中，则调 valueBound()，若从 session 对象的属性中移除，则调 valueUnbound()；实现 HttpSessionBindingListener 接口的类无需在 web.xml 配置

注意：HttpSessionBindingListener 和 HttpSessionListener 之间的最大区别：HttpSessionListener 只需要设置到 web.xml 中就可以监听整个应用中的所有 session。HttpSessionBindingListener 必须实例化后放入某一个 session 中，才可以进行监听。从监听范围上比较，HttpSessionListener 设置一次就可以监听所有 session，HttpSessionBindingListener 通常都是一对一的。

下面的示例实现了 HttpSessionListener 监听器，用于统计网站的在线人数。网站的在线人数往往是一个近似值，统计时一般以 Session 尚未过期的用户人数作为在线人数计算。

```
public class MyHttpSessionListener implements HttpSessionListener {
    public void sessionCreated(HttpSessionEvent event) {
        HttpSession sess = event.getSession();
        ServletContext application = sess.getServletContext();
        Integer online = (Integer)application.getAttribute("online");
        if(online != null)
            online++;
        else
            online = 1;
        application.setAttribute("online", online);
    }
    public void sessionDestroyed(HttpSessionEvent event) {
        HttpSession sess = event.getSession();
        ServletContext application = sess.getServletContext();
        Integer online = (Integer)application.getAttribute("online");
        if(online != null)
            online--;
```

```
            else
                online = 0;
            application.setAttribute("online", online);
        }
    }
```

2.2 通过 web.xml 配置，启用监听器。

在 web.xml 中配置上述的监听器类。

```
<listener>
    <listener-class>com.demo.listener.MyHttpSessionListener</listener-class>
</listener>
```