JSP 表单处理 JSP Cookie 处理

JSP 过滤器

JSP 和 Servlet 中的过滤器都是 Java 类。

过滤器可以动态地拦截请求和响应，以变换或使用包含在请求或响应中的信息。

可以将一个或多个过滤器附加到一个 Servlet 或一组 Servlet。过滤器也可以附加到 JavaServer Pages (JSP) 文件和 HTML 页面。

过滤器是可用于 Servlet 编程的 Java 类，可以实现以下目的：

- 在客户端的请求访问后端资源之前，拦截这些请求。
- 在服务器的响应发送回客户端之前，处理这些响应。

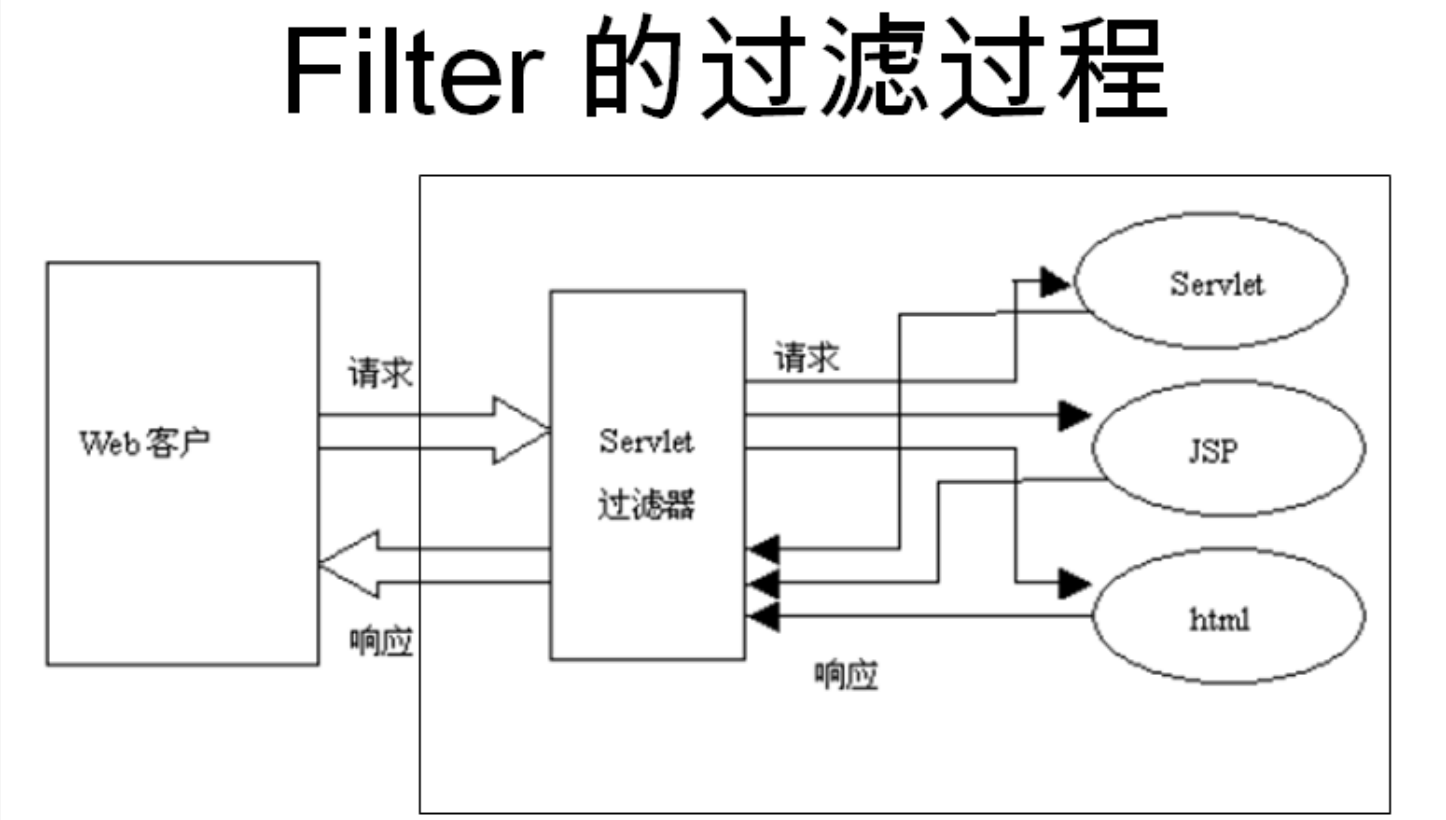
根据规范建议的各种类型的过滤器：

- 身份验证过滤器（Authentication Filters）。
- 数据压缩过滤器（Data compression Filters）。
- 加密过滤器（Encryption Filters）。
- 触发资源访问事件过滤器。
- 图像转换过滤器（Image Conversion Filters）。
- 日志记录和审核过滤器（Logging and Auditing Filters）。
- MIME-TYPE 链过滤器（MIME-TYPE Chain Filters）。
- 标记化过滤器（Tokenizing Filters）。
- XSL/T 过滤器（XSL/T Filters），转换 XML 内容。

过滤器通过 Web 部署描述符（web.xml）中的 XML 标签来声明，然后映射到您的应用程序的部署描述符中的 Servlet 名称或 URL 模式。

当 Web 容器启动 Web 应用程序时，它会为您在部署描述符中声明的每一个过滤器创建一个实例。

Filter 的执行顺序与在 web.xml 配置文件中的配置顺序一致，一般把 Filter 配置在所有的 Servlet 之前。



Servlet 过滤器方法

过滤器是一个实现了 javax.servlet.Filter 接口的 Java 类。javax.servlet.Filter 接口定义了三个方法：

序号	方法 & 描述
1	**public void doFilter (ServletRequest, ServletResponse, FilterChain)** 该方法完成实际的过滤操作，当客户端请求方法与过滤器设置匹配的URL时，Servlet容器将先调用过滤器的doFilter方法。FilterChain用户访问后续过滤器。
2	**public void init(FilterConfig filterConfig)** web 应用程序启动时，web 服务器将创建Filter 的实例对象，并调用其init方法，读取web.xml配置，完成对象的初始化功能，从而为后续的用户请求作好拦截的准备工作（filter对象只会创建一次，init方法也只会执行一次）。开发人员通过init方法的参数，可获得代表当前filter配置信息的FilterConfig对象。
3	**public void destroy()** Servlet容器在销毁过滤器实例前调用该方法，在该方法中释放Servlet过滤器占用的资源。

FilterConfig 使用

Filter 的 init 方法中提供了一个 FilterConfig 对象。

如 web.xml 文件配置如下：

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

```
    <filter-class>com.runoob.test.LogFilter</filter-class>
    <init-param>
        <param-name>Site</param-name>
        <param-value>菜鸟教程</param-value>
    </init-param>
    </filter>
```

在 init 方法使用 FilterConfig 对象获取参数：

```
public void  init(FilterConfig config) throws ServletException {
    // 获取初始化参数
    String site = config.getInitParameter("Site");
    // 输出初始化参数
    System.out.println("网站名称: " + site);
}
```

JSP 过滤器实例

以下是 Servlet 过滤器的实例，将输出网站名称和地址。本实例让您对 Servlet 过滤器有基本的了解，您可以使用相同的概念编写更复杂的过滤器应用程序：

```
//导入必需的 java 库
import javax.servlet.*;
import java.util.*;

//实现 Filter 类
public class LogFilter implements Filter  {
    public void  init(FilterConfig config) throws ServletException {
        // 获取初始化参数
        String site = config.getInitParameter("Site");

        // 输出初始化参数
        System.out.println("网站名称: " + site);
    }
    public void  doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws ja
va.io.IOException, ServletException {

        // 输出站点名称
        System.out.println("站点网址: http://www.runoob.com");

        // 把请求传回过滤链
        chain.doFilter(request,response);
    }
    public void destroy( ){
        /* 在 Filter 实例被 Web 容器从服务移除之前调用 */
```

```
    }
}
```

DisplayHeader.java 文件代码如下：

```
//导入必需的 java 库
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/DisplayHeader")

//扩展 HttpServlet 类
public class DisplayHeader extends HttpServlet {

    // 处理 GET 方法请求的方法
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
 IOException
    {
        // 设置响应内容类型
        response.setContentType("text/html;charset=UTF-8");

        PrintWriter out = response.getWriter();
        String title = "HTTP Header 请求实例 - 菜鸟教程实例";
        String docType =
            "<!DOCTYPE html> \n";
            out.println(docType +
            "<html>\n" +
            "<head><meta charset=\"utf-8\"><title>" + title + "</title></head>\n"+
            "<body bgcolor=\"#f0f0f0\">\n" +
            "<h1 align=\"center\">" + title + "</h1>\n" +
            "<table width=\"100%\" border=\"1\" align=\"center\">\n" +
            "<tr bgcolor=\"#949494\">\n" +
            "<th>Header 名称</th><th>Header 值</th>\n"+
            "</tr>\n");

        Enumeration headerNames = request.getHeaderNames();

        while(headerNames.hasMoreElements()) {
            String paramName = (String)headerNames.nextElement();
            out.print("<tr><td>" + paramName + "</td>\n");
            String paramValue = request.getHeader(paramName);
```

```
            out.println("<td> " + paramValue + "</td></tr>\n");
         }
         out.println("</table>\n</body></html>");
      }
      // 处理 POST 方法请求的方法
      public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException
, IOException {
         doGet(request, response);
      }
}
```

Web.xml 中的 Servlet 过滤器映射（Servlet Filter Mapping）

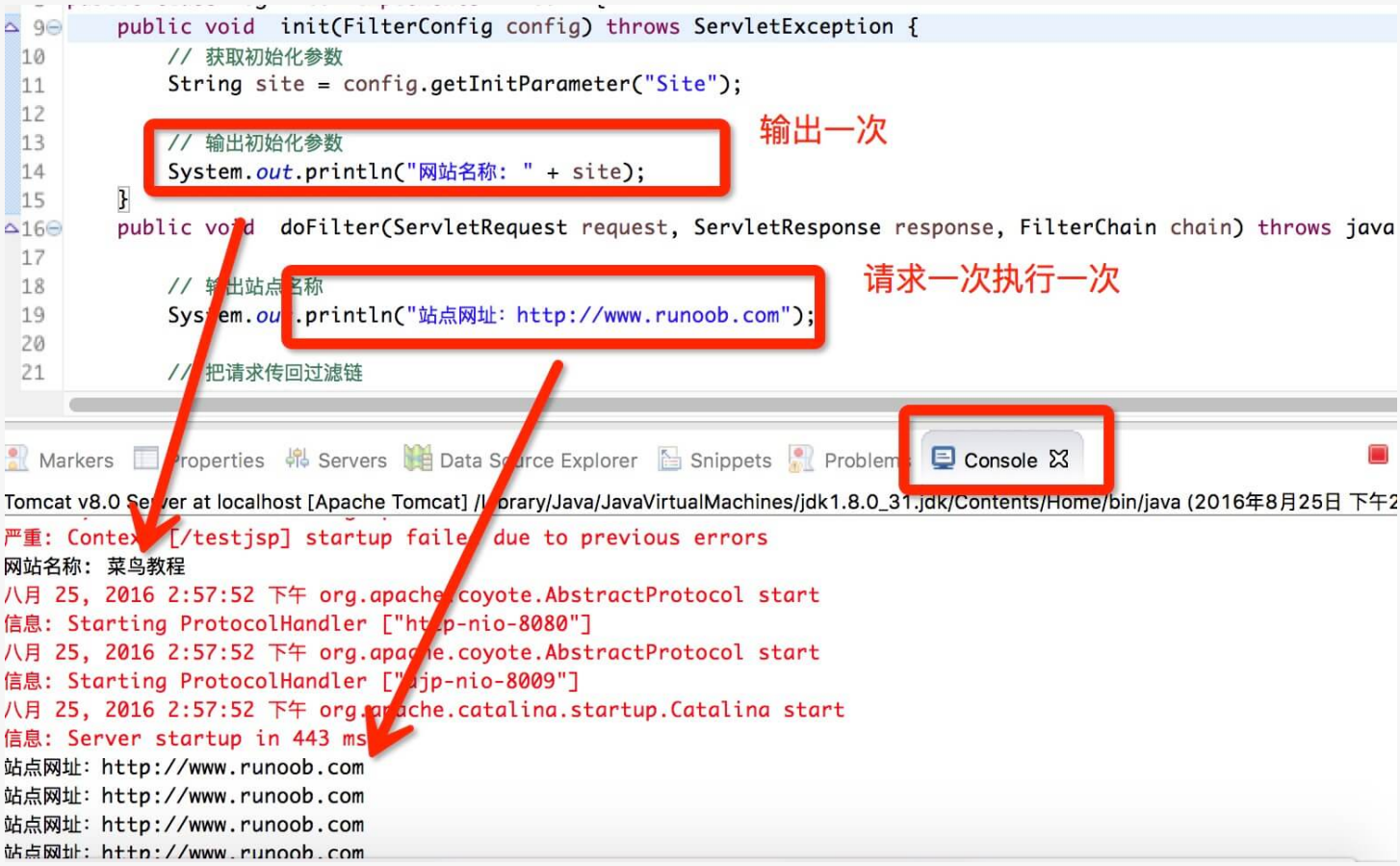
定义过滤器，然后映射到一个 URL 或 Servlet，这与定义 Servlet，然后映射到一个 URL 模式方式大致相同。在部署描述符文件 **web.xml** 中为 filter 标签创建下面的条目：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
<filter>
  <filter-name>LogFilter</filter-name>
  <filter-class>com.runoob.test.LogFilter</filter-class>
  <init-param>
    <param-name>Site</param-name>
    <param-value>菜鸟教程</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<servlet>
  <!-- 类名 -->
  <servlet-name>DisplayHeader</servlet-name>
  <!-- 所在的包 -->
  <servlet-class>com.runoob.test.DisplayHeader</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>DisplayHeader</servlet-name>
  <!-- 访问的网址 -->
  <url-pattern>/TomcatTest/DisplayHeader</url-pattern>
</servlet-mapping>
</web-app>
```

上述过滤器适用于所有的 Servlet，因为我们在配置中指定 */** 。如果您只想在少数的 Servlet 上应用过滤器，您可以指定一个特定的 Servlet 路径。

现在试着以常用的方式调用任何 Servlet，您将会看到在 Web 服务器中生成的日志。您也可以使用 Log4J 记录器来把上面的日志记录到一个单独的文件中。

接下来我们访问这个实例地址 **http://localhost:8080/TomcatTest/DisplayHeader**, 然后在控制台看下输出内容，如下所示：



使用多个过滤器

Web 应用程序可以根据特定的目的定义若干个不同的过滤器。假设您定义了两个过滤器 *AuthenFilter* 和 *LogFilter*。您需要创建一个如下所述的不同的映射，其余的处理与上述所讲解的大致相同：

```
<filter>
   <filter-name>LogFilter</filter-name>
   <filter-class>com.runoob.test.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>com.runoob.test.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>
```

过滤器的应用顺序

web.xml 中的 filter-mapping 元素的顺序决定了 Web 容器应用过滤器到 Servlet 的顺序。若要反转过滤器的顺序，您只需要在 web.xml 文件中反转 filter-mapping 元素即可。

例如，上面的实例将先应用 LogFilter，然后再应用 AuthenFilter，但是下面的实例将颠倒这个顺序：

```
<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>
```

web.xml配置各节点说明

- `<filter>`指定一个过滤器。
  - `<filter-name>`用于为过滤器指定一个名字，该元素的内容不能为空。
  - `<filter-class>`元素用于指定过滤器的完整的限定类名。
  - `<init-param>`元素用于为过滤器指定初始化参数，它的子元素`<param-name>`指定参数的名字，`<param-value>`指定参数的值。
  - 在过滤器中，可以使用`FilterConfig`接口对象来访问初始化参数。
- `<filter-mapping>`元素用于设置一个 Filter 所负责拦截的资源。一个Filter拦截的资源可通过两种方式来指定：Servlet 名称和资源访问的请求路径
  - `<filter-name>`子元素用于设置filter的注册名称。该值必须是在`<filter>`元素中声明过的过滤器的名字
  - `<url-pattern>`设置 filter 所拦截的请求路径(过滤器关联的URL样式)
- `<servlet-name>`指定过滤器所拦截的Servlet名称。

- <dispatcher>指定过滤器所拦截的资源被 Servlet 容器调用的方式，可以是REQUEST,INCLUDE,FORWARD和ERROR之一，默认REQUEST。用户可以设置多个<dispatcher>子元素用来指定 Filter 对资源的多种调用方式进行拦截。
- <dispatcher>子元素可以设置的值及其意义
  - REQUEST：当用户直接访问页面时，Web容器将会调用过滤器。如果目标资源是通过RequestDispatcher的include()或forward()方法访问时，那么该过滤器就不会被调用。
  - INCLUDE：如果目标资源是通过RequestDispatcher的include()方法访问时，那么该过滤器将被调用。除此之外，该过滤器不会被调用。
  - FORWARD：如果目标资源是通过RequestDispatcher的forward()方法访问时，那么该过滤器将被调用，除此之外，该过滤器不会被调用。
  - ERROR：如果目标资源是通过声明式异常处理机制调用时，那么该过滤器将被调用。除此之外，过滤器不会被调用。

← JSP 表单处理 JSP Cookie 处理 →

1 篇笔记 写笔记

一、Filter 的基本工作原理

- 1、Filter 程序是一个实现了特殊接口的 Java 类，与 Servlet 类似，也是由 Servlet 容器进行调用和执行的。
- 2、当在 web.xml 注册了一个 Filter 来对某个 Servlet 程序进行拦截处理时，它可以决定是否将请求继续传递给 Servlet 程序，以及对请求和响应消息是否进行修改。
- 3、当 Servlet 容器开始调用某个 Servlet 程序时，如果发现已经注册了一个 Filter 程序来对该 Servlet 进行拦截，那么容器不再直接调用 Servlet 的 service 方法，而是调用 Filter 的 doFilter 方法，再由 doFilter 方法决定是否去激活 service 方法。
- 4、但在 Filter.doFilter 方法中不能直接调用 Servlet 的 service 方法，而是调用 FilterChain.doFilter 方法来激活目标 Servlet 的 service 方法，FilterChain 对象时通过 Filter.doFilter 方法的参数传递进来的。
- 5、只要在 Filter.doFilter 方法中调用 FilterChain.doFilter 方法的语句前后增加某些程序代码，这样就可以在 Servlet 进行响应前后实现某些特殊功能。
- 6、如果在 Filter.doFilter 方法中没有调用 FilterChain.doFilter 方法，则目标 Servlet 的 service 方法不会被执行，这样通过 Filter 就可以阻止某些非法的访问请求。

更多内容可以参考：Filter、FilterChain、FilterConfig 介绍

lee 9个月前 (06-23)