关于Filter过滤器

笔记本：	关于servlet		
创建时间：	2018/1/26/周五 21:58	**更新时间：**	2018/1/26/周五 22:01
作者：	1634896520@qq.com		
URL：	https://www.cnblogs.com/javabg/p/7092552.html		

关于Filter过滤器

Filter简介

Filter也称之为过滤器，它是Servlet技术中最实用的技术，Web开发人员通过Filter技术，对web服务器管理的所有web资源：例如Jsp, Servlet, 静态图片文件或静态 html 文件等进行拦截，从而实现一些特殊的功能。例如实现URL级别的权限访问控制、过滤敏感词汇、压缩响应信息等一些高级功能。

它主要用于对用户请求进行预处理，也可以对HttpServletResponse进行后处理。使用Filter的完整流程：Filter对用户请求进行预处理，接着将请求交给Servlet进行处理并生成响应，最后Filter再对服务器响应进行后处理。

Filter功能

在HttpServletRequest到达 Servlet 之前，拦截客户的HttpServletRequest 。根据需要检查HttpServletRequest，也可以修改HttpServletRequest 头和数据。

在HttpServletResponse到达客户端之前，拦截HttpServletResponse 。根据需要检查HttpServletResponse，也可以修改HttpServletResponse头和数据。

如何借助Filter实现拦截功能

Filter接口中有一个doFilter方法，当开发人员编写好Filter，并配置对哪个web资源进行拦截后，Web服务器每次在调用web资源的service方法之前，都会先调用一下filter的doFilter方法，因此，在该方法内编写代码可达到如下目的：

调用目标资源之前，让一段代码执行。
是否调用目标资源（即是否让用户访问web资源）。
web服务器在调用doFilter方法时，会传递一个filterChain对象进来，filterChain对象是filter接口中最重要的一个对象，它也提供了一个doFilter方法，开发人员可以根据需求决定是否调用此方法，调用该方法，则web服务器就会调用web资源的service方法，即web资源就会被访问，否则web资源不会被访问。

Filter开发两步走

编写java类实现Filter接口，并实现其doFilter方法。

在web.xml文件中对编写的filter类进行注册，并设置它所能拦截的资源。

web.xml配置各节点介绍：

<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。

Filter链

在一个web应用中，可以开发编写多个Filter，这些Filter组合起来称之为一个Filter链。

web服务器根据Filter在web.xml文件中的注册顺序，决定先调用哪个Filter，当第一个Filter的doFilter方法被调用时，web服务器会创建一个代表Filter链的FilterChain对象传递给该方法。在doFilter方法中，开发人员如果调用了FilterChain对象的doFilter方法，则web服务器会检查FilterChain对象中是否还有filter，如果有，则调用第2个filter，如果没有，则调用目标资源。

Filter的生命周期

public void init(FilterConfig filterConfig) throws ServletException;//初始化
和我们编写的Servlet程序一样，Filter的创建和销毁由WEB服务器负责。 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。

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException;//拦截请求
这个方法完成实际的过滤操作。当客户请求访问与过滤器关联的URL的时候，Servlet过滤器将先执行doFilter方法。FilterChain参数用于访问后续过滤器。

public void destroy();//销毁

Filter对象创建后会驻留在内存，当web应用移除或服务器停止时才销毁。在Web容器卸载 Filter 对象之前被调用。该方法在Filter的生命周期中仅执行一次。在这个方法中，可以释放过滤器使用的资源。

FilterConfig接口

用户在配置filter时，可以使用为filter配置一些初始化参数，当web容器实例化Filter对象，调用其init方法时，会把封装了filter初始化参数的filterConfig对象传递进来。因此开发人员在编写filter时，通过filterConfig对象的方法，就可获得以下内容：

String getFilterName();//得到filter的名称。
String getInitParameter(String name);//返回在部署描述中指定名称的初始化参数的值。如果不存在返回null.
Enumeration getInitParameterNames();//返回过滤器的所有初始化参数的名字的枚举集合。
public ServletContext getServletContext();//返回Servlet上下文对象的引用。

Filter使用案例

使用Filter验证用户登录安全控制

前段时间参与维护一个项目，用户退出系统后，再去地址栏访问历史，根据url，仍然能够进入系统响应页面。我去检查一下发现对请求未进行过滤验证用户登录。添加一个filter搞定问题！

先在web.xml配置

```
<filter>
<filter-name>SessionFilter</filter-name>
<filter-class>com.action.login.SessionFilter</filter-class>
<init-param>
<param-name>logonStrings</param-name><!-- 对登录页面不进行过滤 -->
<param-value>/project/index.jsp;login.do</param-value>
</init-param>
<init-param>
<param-name>includeStrings</param-name><!-- 只对指定过滤参数后缀进行过滤 -->
<param-value>.do;.jsp</param-value>
</init-param>
<init-param>
<param-name>redirectPath</param-name><!-- 未通过跳转到登录界面 -->
<param-value>/index.jsp</param-value>
</init-param>
<init-param>
<param-name>disabletestfilter</param-name><!-- Y:过滤无效 -->
<param-value>N</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>SessionFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

接着编写FilterServlet

```
package com.action.login;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponseWrapper;

/**
* 判断用户是否登录,未登录则退出系统
*/
public class SessionFilter implements Filter {

public FilterConfig config;

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

public static boolean isContains(String container, String[] regx) {
boolean result = false;

for (int i = 0; i < regx.length; i++) {
if (container.indexOf(regx[i]) != -1) {
return true;
}
```

```
}
return result;
}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
HttpServletRequest hrequest = (HttpServletRequest)request;
HttpServletResponseWrapper wrapper = new HttpServletResponseWrapper((HttpServletResponse) response);

String logonStrings = config.getInitParameter("logonStrings"); // 登录登陆页面
String includeStrings = config.getInitParameter("includeStrings"); // 过滤资源后缀参数
String redirectPath = hrequest.getContextPath() + config.getInitParameter("redirectPath");// 没有登陆转向页面
String disabletestfilter = config.getInitParameter("disabletestfilter");// 过滤器是否有效

if (disabletestfilter.toUpperCase().equals("Y")) { // 过滤无效
chain.doFilter(request, response);
return;
}
String[] logonList = logonStrings.split(";");
String[] includeList = includeStrings.split(";");

if (!this.isContains(hrequest.getRequestURI(), includeList)) {// 只对指定过滤参数后缀进行过滤
chain.doFilter(request, response);
return;
}

if (this.isContains(hrequest.getRequestURI(), logonList)) {// 对登录页面不进行过滤
chain.doFilter(request, response);
return;
}

String user = ( String ) hrequest.getSession().getAttribute("useronly");//判断用户是否登录
if (user == null) {
wrapper.sendRedirect(redirectPath);
return;
}else {
chain.doFilter(request, response);
return;
}
}

public void init(FilterConfig filterConfig) throws ServletException {
config = filterConfig;
}
}
```

这样既可完成对用户所有请求，均要经过这个Filter进行验证用户登录。

防止中文乱码过滤器

项目使用spring框架时。当前台JSP页面和Java代码中使用了不同的字符集进行编码的时候就会出现表单提交的数据或者上传/下载中文名称文件出现乱码的问题，那就可以使用这个过滤器。

```
<filter>
<filter-name>encoding</filter-name>
<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
<init-param>
<param-name>encoding</param-name><!--用来指定一个具体的字符集-->
<param-value>UTF-8</param-value>
</init-param>
<init-param>
<param-name>forceEncoding</param-name><!--true：无论request是否指定了字符集，都是用encoding；false：如果request已指定一个字符集，则不使用
encoding-->
<param-value>false</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>encoding</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```