四个有用的Java过滤器

java 代码

## 一、使浏览器不缓存页面的过滤器

**package** com.sist;

**import** javax.servlet.\*;

**import** javax.servlet.http.HttpServletResponse;

**import** java.io.IOException;

/\*\*

\* 用于的使 Browser 不缓存页面的过滤器

\*/

**public** **class** ForceNoCacheFilter **implements** Filter {

**public** **void** doFilter(ServletRequest request, ServletResponse response,

FilterChain filterChain) **throws** IOException, ServletException {

((HttpServletResponse) response).setHeader("Cache-Control", "no-cache");

((HttpServletResponse) response).setHeader("Pragma", "no-cache");

((HttpServletResponse) response).setDateHeader("Expires", -1);

filterChain.doFilter(request, response);

}

}

## 二、检测用户是否登陆的过滤器

**package** com.sist;

**import** javax.servlet.\*;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpSession;

**import** java.util.List;

**import** java.util.ArrayList;

**import** java.util.StringTokenizer;

**import** java.io.IOException;

/\*\*

\* 用于检测用户是否登陆的过滤器，如果未登录，则重定向到指的登录页面

\* 配置参数 checkSessionKey 需检查的在 Session 中保存的关键字

\* redirectURL 如果用户未登录，则重定向到指定的页面，URL不包括 ContextPath

\* notCheckURLList不做检查的URL列表，以分号分开，并且 URL 中不包括 ContextPath

\*/

**public** **class** CheckLoginFilter **implements** Filter {

**private** String redirectURL = **null**;

**private** List notCheckURLList = **new** ArrayList();

**private** String sessionKey = **null**;

**public** **void** doFilter(ServletRequest servletRequest,

ServletResponse servletResponse, FilterChain filterChain)

**throws** IOException, ServletException {

HttpServletRequest request = (HttpServletRequest) servletRequest;

HttpServletResponse response = (HttpServletResponse) servletResponse;

HttpSession session = request.getSession();

**if** (sessionKey == **null**) {

filterChain.doFilter(request, response);

**return**;

}

**if** ((!checkRequestURIIntNotFilterList(request))

&&session.getAttribute(sessionKey) == **null**) {

response.sendRedirect(request.getContextPath() + redirectURL);

**return**;

}

filterChain.doFilter(servletRequest, servletResponse);

}

**public** **void** destroy() {

notCheckURLList.clear();

}

**private** **boolean** checkRequestURIIntNotFilterList(HttpServletRequest request) {

String uri = request.getServletPath()

+ (request.getPathInfo() == **null** ? "" : request.getPathInfo());

**return** notCheckURLList.contains(uri);

}

**public** **void** init(FilterConfig filterConfig) **throws** ServletException {

redirectURL = filterConfig.getInitParameter("redirectURL");

sessionKey = filterConfig.getInitParameter("checkSessionKey");

String notCheckURLListStr = filterConfig

.getInitParameter("notCheckURLList");

**if** (notCheckURLListStr != **null**) {

StringTokenizer st = **new** StringTokenizer(notCheckURLListStr, ";");

notCheckURLList.clear();

**while** (st.hasMoreTokens()) {

notCheckURLList.add(st.nextToken());

}

}

}

}

## 三、字符编码的过滤器

**package** com.sist;

**import** javax.servlet.\*;

**import** java.io.IOException;

/\*\*

\* 用于设置 HTTP 请求字符编码的过滤器，通过过滤器参数encoding指明使用何种字符编码,

\* 用于处理Html Form请求参数的中文问题

\*/

**public** **class** CharacterEncodingFilter **implements** Filter {

**protected** String encoding = "";

**public** **void** doFilter(ServletRequest servletRequest,

ServletResponse servletResponse, FilterChain filterChain)

**throws** IOException, ServletException {

**if** (encoding != **null**)

servletRequest.setCharacterEncoding(encoding);

filterChain.doFilter(servletRequest, servletResponse);

}

**public** **void** destroy() {

encoding = **null**;

}

**public** **void** init(FilterConfig filterConfig) **throws** ServletException {

**this**.encoding = filterConfig.getInitParameter("encoding");

}

}

四、资源保护过滤器

package catalog.view.util;

import javax.servlet.Filter;

import javax.servlet.FilterConfig;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import java.io.IOException;

import java.util.Iterator;

import java.util.Set;

import java.util.HashSet;

//

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

/\*\*

\* This Filter class handle the security of the application.

\*

\* It should be configured inside the web.xml.

\*

\* @author Derek Y. Shen

\*/

public class SecurityFilter implements Filter {

//the login page uri

private static final String LOGIN\_PAGE\_URI = "login.jsf";

//the logger object

private Log logger = LogFactory.getLog(this.getClass());

//a set of restricted resources

private Set restrictedResources;

/\*\*

\* Initializes the Filter.

\*/

public void init(FilterConfig filterConfig) throws ServletException {

this.restrictedResources = new HashSet();

this.restrictedResources.add("/createProduct.jsf");

this.restrictedResources.add("/editProduct.jsf");

this.restrictedResources.add("/productList.jsf");

}

/\*\*

\* Standard doFilter object.

\*/

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

this.logger.debug("doFilter");

String contextPath = ((HttpServletRequest)req).getContextPath();

String requestUri = ((HttpServletRequest)req).getRequestURI();

this.logger.debug("contextPath = " + contextPath);

this.logger.debug("requestUri = " + requestUri);

if (this.contains(requestUri, contextPath) && !this.authorize((HttpServletRequest)req)) {

this.logger.debug("authorization failed");

((HttpServletRequest)req).getRequestDispatcher(LOGIN\_PAGE\_URI).forward(req, res);

}

else {

this.logger.debug("authorization succeeded");

chain.doFilter(req, res);

}

}

public void destroy() {}

private boolean contains(String value, String contextPath) {

Iterator ite = this.restrictedResources.iterator();

while (ite.hasNext()) {

String restrictedResource = (String)ite.next();

if ((contextPath + restrictedResource).equalsIgnoreCase(value)) {

return true;

}

}

return false;

}

private boolean authorize(HttpServletRequest req) {

//处理用户登录

/\* UserBean user = (UserBean)req.getSession().getAttribute(BeanNames.USER\_BEAN);

if (user != null && user.getLoggedIn()) {

//user logged in

return true;

}

else {

return false;

}\*/

}

}

Servlet过滤器大全

四、资源保护过滤器

package catalog.view.util;

import javax.servlet.Filter;

import javax.servlet.FilterConfig;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import java.io.IOException;

import java.util.Iterator;

import java.util.Set;

import java.util.HashSet;

//

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

/\*\* \*//\*\*

\* This Filter class handle the security of the application.

\*

\* It should be configured inside the web.xml.

\*

\* @author Derek Y. Shen

\*/

public class SecurityFilter implements Filter {

//the login page uri

private static final String LOGIN\_PAGE\_URI = "login.jsf";

//the logger object

private Log logger = LogFactory.getLog(this.getClass());

//a set of restricted resources

private Set restrictedResources;

/\*\* \*//\*\*

\* Initializes the Filter.

\*/

public void init(FilterConfig filterConfig) throws ServletException {

this.restrictedResources = new HashSet();

this.restrictedResources.add("/createProduct.jsf");

this.restrictedResources.add("/editProduct.jsf");

this.restrictedResources.add("/productList.jsf");

}

/\*\* \*//\*\*

\* Standard doFilter object.

\*/

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

this.logger.debug("doFilter");

String contextPath = ((HttpServletRequest)req).getContextPath();

String requestUri = ((HttpServletRequest)req).getRequestURI();

this.logger.debug("contextPath = " + contextPath);

this.logger.debug("requestUri = " + requestUri);

if (this.contains(requestUri, contextPath) && !this.authorize((HttpServletRequest)req)) {

this.logger.debug("authorization failed");

((HttpServletRequest)req).getRequestDispatcher(LOGIN\_PAGE\_URI).forward(req, res);

}

else {

this.logger.debug("authorization succeeded");

chain.doFilter(req, res);

}

}

public void destroy() {}

private boolean contains(String value, String contextPath) {

Iterator ite = this.restrictedResources.iterator();

while (ite.hasNext()) {

String restrictedResource = (String)ite.next();

if ((contextPath + restrictedResource).equalsIgnoreCase(value)) {

return true;

}

}

return false;

}

private boolean authorize(HttpServletRequest req) {

//处理用户登录

/\*\*//\* UserBean user = (UserBean)req.getSession().getAttribute(BeanNames.USER\_BEAN);

if (user != null && user.getLoggedIn()) {

//user logged in

return true;

}

else {

return false;

}\*/

}

}

五 利用Filter限制用户浏览权限

在一个系统中通常有多个权限的用户。不同权限用户的可以浏览不同的页面。使用Filter进行判断不仅省下了代码量，而且如果要更改的话只需要在Filter文件里动下就可以。

以下是Filter文件代码：

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;

public class RightFilter implements Filter {

public void destroy() {

}

public void doFilter(ServletRequest sreq, ServletResponse sres, FilterChain arg2) throws IOException, ServletException {

// 获取uri地址

HttpServletRequest request=(HttpServletRequest)sreq;

String uri = request.getRequestURI();

String ctx=request.getContextPath();

uri = uri.substring(ctx.length());

//判断admin级别网页的浏览权限

if(uri.startsWith("/admin")) {

if(request.getSession().getAttribute("admin")==null) {

request.setAttribute("message","您没有这个权限");

request.getRequestDispatcher("/login.jsp").forward(sreq,sres);

return;

}

}

//判断manage级别网页的浏览权限

if(uri.startsWith("/manage")) {

//这里省去

}

}

//下面还可以添加其他的用户权限，省去。

}

public void init(FilterConfig arg0) throws ServletException {

}

}

<!-- 判断页面的访问权限 -->

<filter>

<filter-name>RightFilter</filter-name>

<filter-class>cn.itkui.filter.RightFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>RightFilter</filter-name>

<url-pattern>/admin/\*\*//\*</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>RightFilter</filter-name>

<url-pattern>/manage/\*</url-pattern>

</filter-mapping>

在web.xml中加入Filter的配置，如下：

<filter>

<filter-name>EncodingAndCacheflush</filter-name>

<filter-class>EncodingAndCacheflush</filter-class>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>EncodingAndCacheflush</filter-name>

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

</filter-mapping>

要传递参数的时候最好使用form进行传参，如果使用链接的话当中文字符的时候过滤器转码是不会起作用的，还有就是页面上

form的method也要设置为post，不然过滤器也起不了作用。