Filter

Javaweb的一个重要组件，对请求和响应进行拦截

**基本功能：** 对Servlet容器调用Servlet的过程进行拦截，从而在Servlet进行响应处理的前后实现一些特殊的功能。

**三个接口**： Filter, FilterChain , FilterConfig（没有类可以继承，只能实现接口）

**介绍：**Filter程序就是一个实现Filter接口额Java类。与Servlet程序相似，它由Servlet容器进行调用和执行，需要在web.xml文件中进行注册和设置它所能拦截的资源（Jsp，Servlet，静态图片文本，静态html文件）

1. **创建一个简单的Filter（和Servlet几乎一样）**
2. **创建一个Filter类，实现上Filter接口（类似Servlet）**

**public** **class** HelloFilter **implements** Filter

实现方法：

**public** **void** destroy()，释放资源

**public** **void** init(FilterConfig arg0)，初始化

//FilterConfig和ServletConfig类似

常用方法：

getInitParamenter(String name)(获取配置文件中的属性，参数，属性名)

getServletContext()返回当前放出拦截请求servlet的ServletContext对象

**public** **void** doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)，执行体

//转到下一个过滤器，如果没有下一个过滤器，则转向拦截的资源

//chain代表filter链对象

filterChain.doFilter(servletRequest, servletResponse);

1. **配置xml文件（类似Servlet）**

<filter>

**<!-- 注册名 -->**

<filter-name>HelloFilter</filter-name>

**<!-- 全类名 -->**

<filter-class>filter.HelloFilter</filter-class>

**<!-- filter属性 -->**

<init-param>

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

<param-value>root</param-value>

</init-param>

</filter>

<filter-mapping>

**<!-- 注册名 -->**

<filter-name>HelloFilter</filter-name>

**<!-- 需要拦截的url -->**

<url-pattern>/filter.jsp</url-pattern>

**<!-- 指定过滤器所拦截资源被servlet容器（服务器）调用的方式 默认request-->**

<dispatcher>REQUEST</dispatcher>

<!--

**①request** ：当前用户直接访问页面时（get,post请求直接访问） 服务器将会调用过滤器，如果目标资源是通过RequestDispatcher的include()或forward()方法访问时，那么过滤器不会被调用

②include ：当前目标资源是通过RequestDispatcher的include()方法访问时，过虑器被调用，其他不会被调用

**③forward** ：如果目标资源是通过RequestDispatcher的forward()方法访问时，过虑器被调用，其他不会被调用（<jsp:forword page=”xx”/>和页面声明<@page errorPage=”xx” %>也会被调用）

④error ：如果目标资源被声明为异常处理机制（在xml文件中声明会被触发，在pack中声明不会），被调用时会调用过滤器，其他不会被调用

**可以同时写多个**dispatcher**节点，叠加过滤受理范围**

-->

</filter-mapping>

**注意：如果有多个Filter拦截一个资源，则会按照filter-mapping的顺序拦截。filter和servlet一样是单例的**

1. **Filter的几个典型应用**
2. **使用浏览器不缓存页面的过滤器**

禁用页面缓存的三行代码

response1.setDateHeader("Expires", -1);

response1.setHeader("Cache-Control", "no-cache");

response1.setHeader("Pargma", "no-cache");

（并不是所有浏览器都支持）

1. **字符编码过滤器**

通过encoding指明使用何种字符编码，以处理Html From请求参数的中文问题

* + 1. **写一个filter**

**public** **class** EncodingFilter **extends** HttpFilter{

**private** String encoding;

**protected** **void** init() {

//获取Context中的属性encoding

encoding=getFilterConfig().getServletContext().getInitParameter("encoding");

}

@Override

**public** **void** doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

**throws** IOException, ServletException {

request.setCharacterEncoding(encoding);

chain.doFilter(request, response);

}

}

* + 1. **编写xml文件**

添加一个属性

<context-param>

<!-- 编码写在这，修改时只需要修改配置文件，无需修改源码 -->

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

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

</context-param>

所有的servlet都被此过滤器过滤

<filter>

<filter-name>EncodingFilter</filter-name>

<filter-class>filter.EncodingFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>EncodingFilter</filter-name>

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

</filter-mapping>

1. **检测用户是否登录的过滤器**

登录之前只能访问列表中的a页面，访问其他页面跳转到登录页面，登录后可以访问所有页面

关键部分有两个

* 1. 添加到xml三个context属性

<!-- 用于确定保存登录时输入的session属性名 -->

<context-param>

<param-name>userSessionKey</param-name>

<param-value>USERSESSIONKEY</param-value>

</context-param>

<!-- 如果未登录，重定向的页面 -->

<context-param>

<param-name>rediretPage</param-name>

<param-value>/test/filtercheck/login.jsp</param-value>

</context-param>

<!-- 不需要拦截的页面,逗号分隔 -->

<context-param>

<param-name>uncheckedUrls</param-name>

<param-value>/test/filtercheck/login.jsp,/test/filtercheck/dologin.jsp,/test/filtercheck/a.jsp,/test/filtercheck/list.jsp</param-value>

</context-param>

* 1. 创建一个filter

1. **初始化获取三个context属性**

ServletContext context=getFilterConfig().getServletContext();

//获取wen.xml中的uerSessionKey值

user=context.getInitParameter("userSessionKey");

//获取uncheckUrls值，不需要拦截的url，以逗号分隔

uncheckedUrls=context.getInitParameter("uncheckedUrls").split(",");

//获取重定向的路径

path=context.getInitParameter("rediretPage");

1. **编写过滤执行部分**

// 判断当前过滤的路径是否是不需要拦截的路径

List<String> urls = Arrays.*asList*(uncheckedUrls);

**if** (urls.contains(request.getServletPath()))

**return**;

// 检测是否已登录，如果未登录则重定向到login.jsp，如果已登录则打开页面（事先要在登录时把登录名存入session中）

HttpSession session = request.getSession();

**if** (session.getAttribute(user) == **null**) {

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

} **else** {

chain.doFilter(request, response);

}

1. **高级应用**
2. **补充**

**Request获取路径的方法**

getRequestURI():/NewjavaWeb/filterPermission/page\_1.jsp

站点路径

getServletPath():/filterPermission/page\_1.jsp

web应用下路径

两个实现包装类：

HttpServletRequest接口的实现类

HttpServletRequestWrapper

HttpServletResponse接口的实现类

HttpServletResponseWrapper

用于通过继承方式增强服务器提供实现类的方法，

（如：编写MyHttpServletRequest继承于实现类，重写getParment方法，改变其行为，过滤获取到的信息）（使用时传入使用的request，实际操作的还是这个request）