**过滤器(Filter)**

**什么是Filter：**

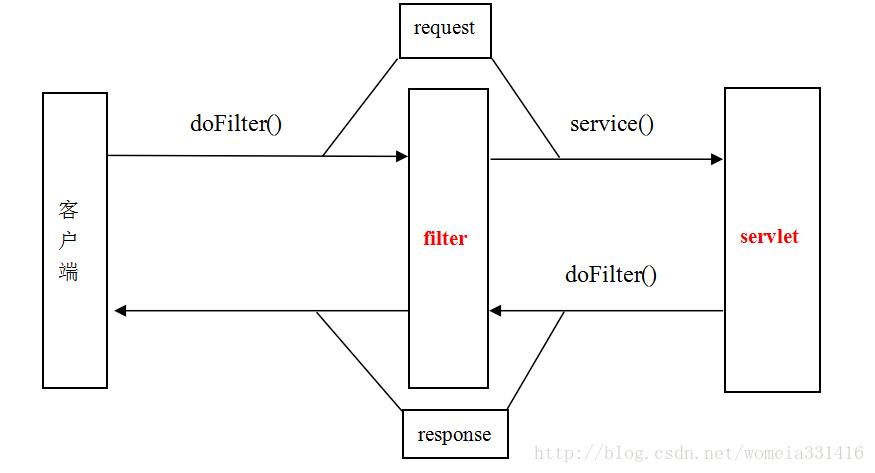
过滤器是用于过滤Servlet的请求和响应，过滤器是在客户端和Servlet之间。当客户端请求某一个Servlet的时候，请求会先进入Filter的doFilter(),开始处理,处理完成以后,调用chain.doFilter(request, response)把request,response继续传递,让请求继续往下走,进入Servlet,在Servlet处理完成以后,又重回Filter的doFilter(),执行chain.doFilter(request, response)下面的语句,完成以后，响应给客户端.

【注意】：jsp文件也是可以过滤的，因为jsp文件的本质也是servlet文件。

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException{

}  
  
【注意】：Filter不会产生request和response对象,只是对request和response的值进行修改，起到一个过滤的作用

**原理图**



**filter在xml中的配置**

<servlet>

<servlet-name>MyServlet</servlet-name>

<servlet-class>com.accp.servlet.MyServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>MyServlet</servlet-name>

<url-pattern>/MyServlet</url-pattern>

</servlet-mapping>

<filter> <!--过滤器的类 -->

<filter-name>MyFilter</filter-name>

<filter-class>com.accp.filter.MyFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>MyFilter</filter-name>

<servlet-name>MyServlet</servlet-name> <!-- 表示只过滤MyServlet这一个servlet,其他Servlet不管 -->

<url-pattern>/\*</url-pattern><!-- 表示所有的请求都被过滤-->

</filter-mapping>

1、过滤器必须实现filter接口(标准)

2、filter声明周期：

在什么时候构造？ 在服务器启动，项目部署完。

在什么时候初始化？在服务器启动，项目部署完。

在什么时候进行过滤？在客户端访问servlet的时候。

在什么时候效率？ 项目移除或重新部署，服务器关闭

3、filter生命周期与servlet的生命周期对比

构造和初始化的时期不同：

filter 实在服务器启动，项目部署

servlet 在第一次请求是进行构造

【注意】filter是服务器启动就产生了,并且进行初始化,而servlet一定是有了请求才会出生和初 始化，从打印的结果可以看到请求先进入filter，回去的时候又从过滤器经过. 当服务器重启或者关闭，servlet先死，filter后死

**Filter应用**

# 一：解决中文乱码问题（request和response）

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest)req;

HttpServletResponse response = (HttpServletResponse)res;

if(request.getMethod().equalsIgnoreCase("post")){

request.setCharacterEncoding("gb18030");

}else{

Enumeration enum = request.getParameterNames();

while(enum.hasMoreElements()){

String key = (String)enum.nextElement();

String[] values = request.getParameterValues(key);

for(int i=0;i<values.length;i++){

values[i]=newString(values[i].getBytes("iso8859-1"),"gb18030");

}

}

} //必须写在过滤之前，写在之后就会失效

response.setContentType("text/html;charset=gb18030");

chain.doFilter(request, response);

}

二：解决权限问题

public void doFilter(ServletRequest req, ServletResponse resp,

FilterChain chain) throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest)req;

HttpServletResponse response = (HttpServletResponse)resp;

//得到session的值

String role = (String)request.getSession().getAttribute("role");

//拿到项目的路径，相当于"/项目名"

System.out.println("-------------"+request.getContextPath());

if(role==null || !role.equals("admin")){

response.sendRedirect(request.getContextPath()+"/error.jsp");

}

chain.doFilter(request, response);

}

三：过滤掉非法文字过滤掉非法文字

String方法

indexOf()查询参数字符串是否在字符串中的下标

String a="日本人";

a.indexOf()

a.准备一个str.properties文件，把要过滤掉的字符，准备在里面

\u738b\u516b=\*

\u65e5\u672c=\*\*

\u6740\u624b=\*

注意这里要转码

\u738b\u516b–>王八

\u65e5\u672c–>日本

\u6740\u624b–>杀手

**b.**过滤器代码

Properties ps = null;

publicvoid doFilter(ServletRequest req, ServletResponse resp,FilterChain chain) throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest)req;

HttpServletResponse response = (HttpServletResponse)resp;

//注意中文乱码的设置

request.setCharacterEncoding("gb18030");

Enumeration enum = request.getParameterNames();

while(enum.hasMoreElements()){

String key = (String)enum.nextElement();

String[] values = request.getParameterValues(key);

for (int i = 0; i < values.length; i++) {

//根据key,从properties中拿到值,如果不为空,则证明一定有非法字符

//然后替换掉

if(ps.get(values[i])!=null){

values[i] = ps.getProperty(values[i]);

}

}

}

chain.doFilter(request, response);

}

//初始化中读取str.properties文件

publicvoid init(FilterConfig filterConfig) throws ServletException {

try {

//读取str.properties文件

InputStream in = filterConfig.getServletContext().getResourceAsStream("/WEB-

INF/str.properties");

ps = new Properties();

ps.load(in);//加载配置文件

in.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

四：dispatcher的作用

2.4版本的servlet规范在部属描述符中新增加了一个元素，这个元素有四个可能的值：即REQUEST,FORWARD,INCLUDE和ERROR，可以在一个元素中加入任意数目的，使得filter将会作用于直接从客户端过来的request，通过forward过来的request，通过include过来的request和通过过来的request。如果没有指定任何< dispatcher >元素，默认值是REQUEST。可以通过下面几个例子来辅助理解。

【注意：】dispacher的配置值中 ：默认是：REQUEST

REQUEST:直接从浏览器过来的请求都会过滤过来   
 FORWARD:通过转发请求指定资源   
 INCLUDE：通过包含的方式的请求(静态包含不包括)   
 ERROR：如果请求是通过发起的 就过滤

Xml代码:

<filter-mapping>

<filter-name>Logging Filter</filter-name>

<servlet-name>ProductServlet</servlet-name>

<dispatcher>INCLUDE</dispatcher>

</filter-mapping>

这种情况下，如果请求是通过request dispatcher的include方法传递过来的对ProductServlet的请求，则要经过这个过滤器的过滤。其它的诸如从客户端直接过来的对ProductServlet的请求等都不需要经过这个过滤器。

指定filter的匹配方式有两种方法：直接指定url-pattern和指定servlet，后者相当于把指定的servlet对应的url-pattern作为filter的匹配模式

filter的路径匹配和servlet是一样的

Xml代码

<filter-mapping>

<filter-name>Logging Filter</filter-name>

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

<dispatcher>FORWARD</dispatcher>

<dispatcher>REQUEST</dispatcher>

</filter-mapping>

在这种情况下，如果请求是以/products/…开头的，并且是通过request dispatcher的forward方法传递过来或者直接从客户端传递过来的，则必须经过这个过滤器。