

## 一、定义

1 | 当访问服务器资源时，过滤器可以将请求拦截下来，完成一些特殊的功能。

1 | 过滤器常见字符串：过滤一些敏感字符串、避免中文乱码【在过滤器中规定编码方式，规定web资源都使用UTF-8编码】、权限验证【规定只有带Session或Cookie的浏览器，才能访问web资源】等等等。过滤器的作用非常强大，只要发挥想象就可以有意想不到的效果。

## 二、Filter接口方法

1 |  
2 | 1、init：在服务器启动后，会创建Filter对象，然后调用init方法，只执行一次，用于加载资源。  
3 |  
4 | 2、doFilter：Filter的核心方法，用于执行Filter。每一次要请求被拦截的资源时，就会执行，可以执行多次。  
5 |  
6 | 3、destroy：在服务器关闭后，Filter对象被销毁。如果服务器正常关闭，则会执行destroy方法，只执行一次，用于释放资源。  
7 |

## 三、操作步骤

### 1、定义一个类实现Filter接口

1 | 注意：Filter接口的包为javax.servlet.\*

### 2、复写Filter接口方法

### 3、配置拦截路径

1 | 意思是客户端要访问的资源若是拦截路径里填写的，那过滤器就会生效

### 4、决定是否放行

### 5、具体实现

实现Filter的类，通过filterChain的doFilter方法决定是否放行，如果不放行，则客户端访问的资源访问不到。如果放行，则客户端可以访问到资源。

#### ①未放行

```

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;
@WebFilter("/index.jsp")// 若客户端访问的是index.jsp的资源，则会执行此过滤器
public class FilterDemo1 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException {
        System.out.println("FilterDemo1被访问了");





        /*我们执行过滤器后要考虑是否放行,通过filterChain调用doFilter, 传入request与response,
        这行代码作用就是放行,如果没有这行代码, 客户端要访问的资源就被拦截。*/
        //filterChain.doFilter(servletRequest,servletResponse);
    }

    @Override
    public void destroy() {
    }
}

```

FilterDemo1被访问了

1 | 因为FilterDemo1没有放行，所以访问不到index.jsp的资源，结果如下：

 应用
  译
  百度翻译
  Home - Quora

## ②放行后

```

@WebFilter("/index.jsp")// 若客户端访问的是index.jsp的资源，则会执行此过滤器
public class FilterDemo1 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException {
        System.out.println("FilterDemo1被访问了");

        /*我们执行过滤器后要考虑是否放行,通过filterChain调用doFilter, 传入request与response,
        这行代码作用就是放行,如果没有这行代码, 客户端要访问的资源就被拦截。*/
        filterChain.doFilter(servletRequest,servletResponse);
    }

    @Override
    public void destroy() {
    }
}

```

因为放行了，所以index.jsp资源被访问到了。

FilterDemo1被访问了

## 通过过滤器FilterDemo1放行，我被执行了

### 四、Filter配置方式

#### 1、拦截路径多种写法

##### (1) 具体资源路径

1 | 如拦截路径为:/index.jsp,则客户端要访问index.jsp资源时，就要执行过滤器

##### (2) 拦截目录

1 | 如拦截路径为: /user/\* ，则客户端如果要访问user下的资源时，执行该过滤器。

①定义一个Filter.拦截路径为/User/\*，意为拦截此路径下的所有资源。

```
@WebFilter("/User/*")
public class ToBindFilter implements Filter {
    public void destroy() {
    }

    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException {
        System.out.println("准备放行");
        chain.doFilter(req, resp);
    }
}
```

②创建两个User目录下的Servlet.

```
@WebServlet("/User/ServletDemo1")
public class ServletDemo1 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("过滤器放行执行到我，我是ServletDemo1");
    }
}
```

```
@WebServlet("/User/ServletDemo2")
public class ServletDemo2 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("过滤器放行执行到我，我是ServletDemo2");
    }
}
```

③结果

localhost/Wu/User/ServletDemo1|

```
Output
↑ 准备放行
↓ 过滤器放行执行到我，我是ServletDemo1

localhost/Wu/User/ServletDemo2

准备放行
过滤器放行执行到我，我是ServletDemo2
```

### (3) 后缀名拦截

1 | 如拦截路径为: \*.jsp:如果客户端要访问后缀名为jsp的资源时，则执行过滤器

### (4) 全资源拦截

1 | 如: /\*: 客户端访问资源时，就会执行该过滤器。

1 | 配置方式有两种: XML配置法和web.xml配置法

## 2、XML配置法

1 | 在web.xml文件中配置，需要填写的参数暂时用xxx代替。

```
1  <filter>
2    <!--用于为过滤器指定一个名字，该元素的内容不能为空。-->
3    <filter-name>xxx</filter-name>
4
5    <!--元素用于指定过滤器的完整的全类名-->
6    <filter-class>xxx</filter-class>
7
8  </filter>
9
10
11  <filter-mapping>
12
13    <!--设置filter的名称，和上面一样的名字-->
14    <filter-name>xxx</filter-name>
15
16    <!--设置 filter 所拦截的请求路径。-->
17    <url-pattern>xxx</url-pattern>
18
19  </filter-mapping>
```

## (1) 具体演示

### ①web.xml内

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://
  version="4.0">

  <filter>
    <filter-name>Filter</filter-name>
    <filter-class>BaseFilter.FilterDemo1</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>Filter</filter-name>
    <url-pattern>/index.jsp</url-pattern>
  </filter-mapping>
</web-app>
```

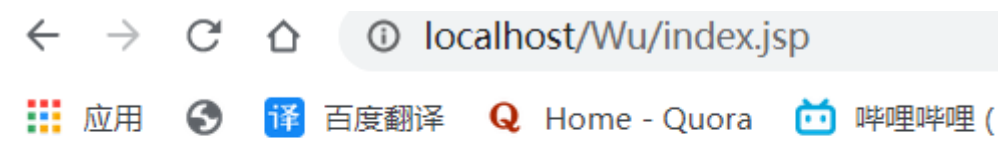
### ②Filter内

```
import java.io.IOException;
public class FilterDemo1 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

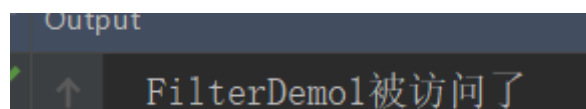
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)
        throws IOException, ServletException {
        System.out.println("FilterDemo1被访问了");
        filterChain.doFilter(servletRequest, servletResponse);
    }

    @Override
    public void destroy() {
    }
}
```

### ③结果



通过过滤器FilterDemo1放行，我被执行了



### 3、注解配置法

1 | 即在实现Filter接口的类上添加一个@WebFilter(“拦截路径”)。

### 4、过滤器拦截方式配置

#### (1) 四种拦截方式

##### ①REQUEST

1 | 默认值，客户端直接访问资源时，过滤器生效。

##### ②FORWARD

1 | 当客户端直接访问资源时，过滤器不生效，只有该资源转发给另一个资源时，过滤器才生效。

##### ③INCLUDE

1 | 包含访问资源（了解）

##### ④ERROR

1 | 跳转错误页面，执行过滤器（了解）

#### (2) 注解设置拦截方式

1 | 通过dispatcherTypes设置属性

演示

```
import java.io.IOException;

@WebFilter(value="/ForwardServlet",dispatcherTypes={DispatcherType.FORWARD})
public class FilterDemol implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse) throws IOException {
        System.out.println("开始放行");
        filterChain.doFilter(servletRequest, servletResponse);
        System.out.println("回来了");
    }
}
```

```

@WebServlet("/TestServlet")
public class TestServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("我是TestServlet, 由于filter现在设置了拦截方式只有转发才生效, 所以来试下转发");
        request.getRequestDispatcher("/ForwardServlet").forward(request, response);
    }
}

```

```

@WebServlet("/ForwardServlet")
public class ForwardServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("TestServlet转发到我这里, 过滤器开始生效, 放行后执行到我这里的代码了");
    }
}

```

localhost/Wu/TestServlet

Output

```

↑ 我是TestServlet, 由于filter现在设置了拦截方式只有转发才生效, 所以来试下转发
↓ 开始放行
⌕ TestServlet转发到我这里, 过滤器开始生效, 放行后执行到我这里的代码了
↕ 回来了
🖨

```

1 | 未经过过滤器情况, 直接访问ForwardServlet, 而不是转发。

localhost/Wu/ForwardServlet

因为过滤器设置方式为转发拦截, 而现在我是被直接访问, 所以没执行过滤器

### 思考题

1 | 如果FilterDemo1注解的拦截路径设置为/\*, 且拦截方式为FORWARD和REQUEST, 那么Filter会被执行几次? 答: 两次

```

import java.io.IOException;

@WebFilter(value="/*", dispatcherTypes = {DispatcherType.FORWARD, DispatcherType.REQUEST})
public class FilterDemo1 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        System.out.println("开始放行");
        filterChain.doFilter(servletRequest, servletResponse);
        System.out.println("回来了");
    }
}

```

```
@WebServlet("/TestServlet")
public class TestServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException {
        System.out.println("TestServlet....");
        request.getRequestDispatcher(s: "/ForwardServlet").forward(request, response);
    }
}
```

```
@WebServlet("/ForwardServlet")
public class ForwardServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException {
        System.out.println("TestServlet转发到我这里来");
    }
}
```

localhost/Wu/TestServlet

```
开始放行
TestServlet....
开始放行
TestServlet转发到我这里来
回来了
回来了
```

### (3) web.xml拦截方式

- 1 | 配置在<filter-mapping>的<url-pattern>下，把属性配置在<dispatcher>标签里即可

```
1 | <filter-mapping>
2 |     <filter-name>myfilter</filter-name>
3 |     <url-pattern>/test.jsp</url-pattern>
4 |     <dispatcher>REQUEST</dispatcher>
5 |     <dispatcher>FORWARD</dispatcher>
6 | </filter-mapping>
```

## 五、执行流程

### 1、执行过滤器

### 2、执行放行后的资源

### 3、资源执行完成后，回到过滤器执行放行代码下边的代码

- 1 | 完整执行流程：当服务器收到客户端请求后，如果有过滤器，则会通过过滤器过滤掉一些东西或增强请求，然后放行后，如果还有过滤器拦截，则进入过滤器。如果没有，则Servlet执行，等Servlet执行完后，还需要经过过滤器处理，处理完响应给客户端。



```

@WebFilter("/*")
public class FilterDemo2 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        System.out.println("准备放行");
        //执行这一句，说明放行（让下一个过滤器执行，或者执行目标资源）
        filterChain.doFilter(servletRequest, servletResponse);
        System.out.println("目标资源执行完后，回到此过滤器来，继续执行代码");
    }
}

```

```

@WebServlet("/TestServlet")
public class TestServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("我是TestServlet, filter放行后，执行我里面的代码。");
    }
}

```

localhost/Wu/TestServlet

```

Output
准备放行
我是TestServlet, filter放行后，执行我里面的代码。
目标资源执行完后，回到此过滤器来，继续执行代码

```

## 六、多个Filter(Filter链)执行顺序

### 1、执行顺序

若有两过滤器，过滤器1和过滤器2

- (1) 执行过滤器1
- (2) 执行过滤器2
- (3) 执行资源
- (4) 回到过滤器2
- (5) 回到过滤器1

### 2、过滤器先后顺序问题

#### (1) 注解配置

1 | 按照类名的字符串比较规则去比较，值小的先执行。

1 | 例子：如AFilter和BFilter，过滤器类名，由字符串比较规则比，第一个字符和第二个字符进行比较A比B小，所以AFilter就先执行了。

```

@WebFilter("/*")
public class FilterDemo1 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws ServletException, IOException {
        System.out.println("FilterDemo1开始执行");
        filterChain.doFilter(servletRequest, servletResponse);
        System.out.println("回到了FilterDemo1, 继续执行FilterDemo1代码");
    }
}

```

```

@WebFilter("/*")
public class FilterDemo2 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws ServletException, IOException {
        System.out.println("FilterDemo2开始执行");
        filterChain.doFilter(servletRequest, servletResponse);
        System.out.println("回到了FilterDemo2, 继续执行FilterDemo2代码");
    }
}

```

```

@WebServlet("/TestServlet")
public class TestServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("我是TestServlet, filter放行后, 执行我里面的代码。");
    }
}

```

localhost/Wu/TestServlet

Server Tomcat Localhost Log Tomcat Catalina Log

tput

```

FilterDemo1开始执行
FilterDemo2开始执行
我是TestServlet, filter放行后, 执行我里面的代码
回到了FilterDemo2, 继续执行FilterDemo2代码
回到了FilterDemo1, 继续执行FilterDemo1代码

```

## (2) web.xml配置

```
version="4.0" />

<filter>
    <filter-name>FilterDemo1</filter-name>
    <filter-class>BaseFilter.FilterDemo1</filter-class>
</filter>
<filter>
    <filter-name>FilterDemo2</filter-name>
    <filter-class>BaseFilter.FilterDemo2</filter-class>
</filter>
<filter-mapping>
    <filter-name>FilterDemo2</filter-name>
    <url-pattern>/TestServlet</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>FilterDemo1</filter-name>
    <url-pattern>/TestServlet</url-pattern>
</filter-mapping>
```

FilterDemo2的filter-mapping标签在FilterDemo1上，所以先执行

```
import java.io.IOException;

public class FilterDemo2 implements Filter {
    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
        System.out.println("FilterDemo2比FilterDemo1先开始执行，因为注解里我的filter-mapping在它上面");
        filterChain.doFilter(servletRequest, servletResponse);
        System.out.println("回到了FilterDemo2，继续执行FilterDemo2代码");
    }
}
```

FilterDemo1:

```
public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {
    System.out.println("FilterDemo1开始执行");
    filterChain.doFilter(servletRequest, servletResponse);
    System.out.println("回到了FilterDemo1，继续执行FilterDemo1代码");
}
```

```
Servlet Log
Tomcat-Localhost Log
Tomcat-Cluster Log
Output

FilterDemo2比FilterDemo1先开始执行，因为注解里我的filter-mapping在它上面
FilterDemo1开始执行
我是TestServlet，filter放行后，执行我里面的代码。
回到了FilterDemo1，继续执行FilterDemo1代码
回到了FilterDemo2，继续执行FilterDemo2代码
```

## 七、过滤器的应用

### 1、登录验证

1	需求
2	
3	(1) 对之前做过的登录查询案例进行改造，访问时，验证其是否登录
4	
5	(2) 如果登录了，则直接放行
6	
7	(3) 如果没有登录，则跳转到登录界面，提示"你尚未登录，请先登录。"

```
@WebFilter("/*")
public class LoginFilter implements Filter {
    public void destroy() {}

    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException {
        /*由于ServletRequest是HttpServletRequest的父类，且对象实例化是由其子类HttpServletRequest实现，所以可以向下
        转型，以便调用HttpServletRequest里独有的方法。*/
        HttpServletRequest request=(HttpServletRequest)req;
        /*与上同理*/
        HttpServletResponse response=(HttpServletResponse)resp;
        //获取客户端要访问的资源路径
        String requestURL = request.getRequestURL().toString();
        //将css、js、fonts、login.jsp、LoginServlet.jsp、YzmServlet等资源排除掉，当访问他们时，直接放行。/css/意思是css目录下的资源
        if(requestURL.contains("/login.jsp")||requestURL.contains("/LoginServlet")||requestURL.contains("/YzmServlet")||
        requestURL.contains("/css/")||requestURL.contains("/js/")||requestURL.contains("/fonts/")){
            chain.doFilter(req, resp);
        }else{
            /*通过session来判断是否登录，如果未登录，则获取到的属性loginUser的值应为null，则不能放行，
            而是转发到login.jsp页面，让其登录。如果不为null，则说明已经登录过了，就让其直接放行。
            */
            if(request.getSession().getAttribute(s: "loginUser")!=null){
                chain.doFilter(req, resp);
            }else{
                request.getSession().setAttribute(s: "tips", o: "用户尚未登录，请登录");
                request.getRequestDispatcher(s: "/login.jsp").forward(request,response);
            }
        }
    }
}
```

### 2、用来设置整个应用的字符编码

### 3、敏感词汇过滤

```
1 import jdk.nashorn.internal.ir.WhileNode;
2
3 import javax.servlet.*;
4 import javax.servlet.annotation.WebFilter;
```

```

5  import java.io.*;
6  import java.lang.reflect.InvocationHandler;
7  import java.lang.reflect.Method;
8  import java.lang.reflect.Proxy;
9  import java.util.ArrayList;
10 import java.util.List;
11 import java.util.Map;
12 import java.util.Set;
13
14 @WebFilter("/*")
15 public class SensitiveStringFilter implements Filter {
16     public void destroy() {
17     }
18
19     //定义一个List集合存储IO流读取到的字符串。
20     private List<String> list = new ArrayList();
21
22     public void doFilter(ServletRequest req, ServletResponse resp,
23 FilterChain chain) throws ServletException, IOException {
24         //生成代理对象
25         ServletRequest request = (ServletRequest)
26 Proxy.newProxyInstance(req.getClass().getClassLoader(),
27 req.getClass().getInterfaces(), new InvocationHandler() {
28
29             @Override
30             public Object invoke(Object proxy, Method method, Object[] args)
31             throws Throwable {
32                 //判断是否是getParameter方法，是则进入
33                 if (method.getName().equals("getParameter")) {
34                     String value = (String) method.invoke(req, args);
35                     //替换字符串。
36                     if (list != null) {
37                         for (String str : list) {
38                             if (value.contains(str)) {
39                                 value = value.replace(str, "*****");
40                             }
41                         }
42                     }
43                     return value;
44                 }
45
46                 //判断是否是getParameterMap，由于getParameterMap只能读不能写的原因，
47                 //不知道怎么过滤
48
49                 //判断是否是getParameterValues。
50                 return method.invoke(req, args);
51             }
52         });
53
54         //将代理对象传到Servlet去
55         chain.doFilter(request, resp);
56     }
57
58     public void init(FilterConfig config) throws ServletException {
59         ServletContext context = config.getServletContext();
60         //获取真实路径
61         String realPath = context.getRealPath("/WEB-INF/classes/敏感词
62 汇.txt");
63         try {
64             //对字符流进行编码设置

```

```
56         InputStreamReader is = new InputStreamReader(new
FileInputStream(realPath), "UTF-8");
57         BufferedReader br = new BufferedReader(is);
58         String str = null;
59         while ((str = br.readLine()) != null) {
60             list.add(str);
61         }
62
63         br.close();
64     } catch (Exception e) {
65         e.printStackTrace();
66     }
67
68 }
69
70 }
71
```