# 过滤器

# 学习目标

1. 能够说出过滤器的作用
2. 能够编写过滤器
3. 能够说出过滤器生命周期相关方法
4. 能够根据过滤路径判断指定的过滤器是否起作用
5. 能够说出什么是过滤器链
6. 能够编写过滤器解决全局乱码

反馈

|  |
| --- |
| 案例有种无从下手的感觉  从请求分析到最后响应，理出自己的思路 |
| 三层架构概念还是很模糊，各种层,好混乱啊...  Mvc+三层架构  Web层  Servlet,控制器，调度所有进行处理  异常在这里处理，给友好提示，服务器正忙。。。请明天再试。。。  Jsp，视图，显示动态数据的  Service层  业务逻辑层，处理业务的  检查用户名是否被注册  调用dao数据库添加用户  Dao层  操作数据库（定义sql,执行sql）,返回最原始的数据  Model---javaBean封装与传递数据 |
| 希望笔记做详细一点 |
| JSP里面有定义好的对象是吧?怎么解决转发乱码的问题?  Jsp默认只是解决输出数据的乱码，没有解决请求乱码，解决请求乱码还是使用request.setCharacterEncoding(“utf-8”); |
| 做案例的时候 老是一个类跳一个类 跳来跳去都懵逼了 要怎么办  跳转页面  前端跳  A标签  Form提交  Location.href  history.go(-1),访问缓存页面  后端跳  Request请求转发  Response重定向（前端跳）  Refresh,定时跳转，3；url=https://www.baidu.com(前端跳) |
| JSP就讲完了吗，感觉没有学到什么，好像很多企业找人都要会jsp的  Jsp显示布局动态数据的， |
| 通常刷新页面都是使用js的history对象刷新页面的方法，还是使用Servlet的refresh响应头来做刷新页面的功能?  建议使用前端，refresh为了后面面试用。  jsp怎么获得cookie数据?  ${cookie.cookie的name.value},一般不在jsp上获取cookie. |
| 1 jsp中,out=pageContext.getOut(), out.print() 跟out.writer()的用法一样吗? pageContext指的是对当前页面有效吗?  ,out=pageContext.getOut(), out.print() 跟out.writer(),服务器将jsp翻译的源代码  pageContext，页面域对象，只能在当前页面用。  Out输出对象。  out.print(object)  跟out.writer(字符串),  2 var：在不循环对象的时候，保存的是控制循环的变量.怎么体现的?  <c:forEach begin=”1” end=”5” step=”1” var=”i”>  Hello world,${i}  </c:forEach> |
| request、ServletContext这两个域对象都有getContextPath方法，他们之间有区别吗？  一样的 |
| 您要不还是讲解一下联系人的增删改怎么实现呗  今天最后的结束的时候。 |

# 过滤器简介

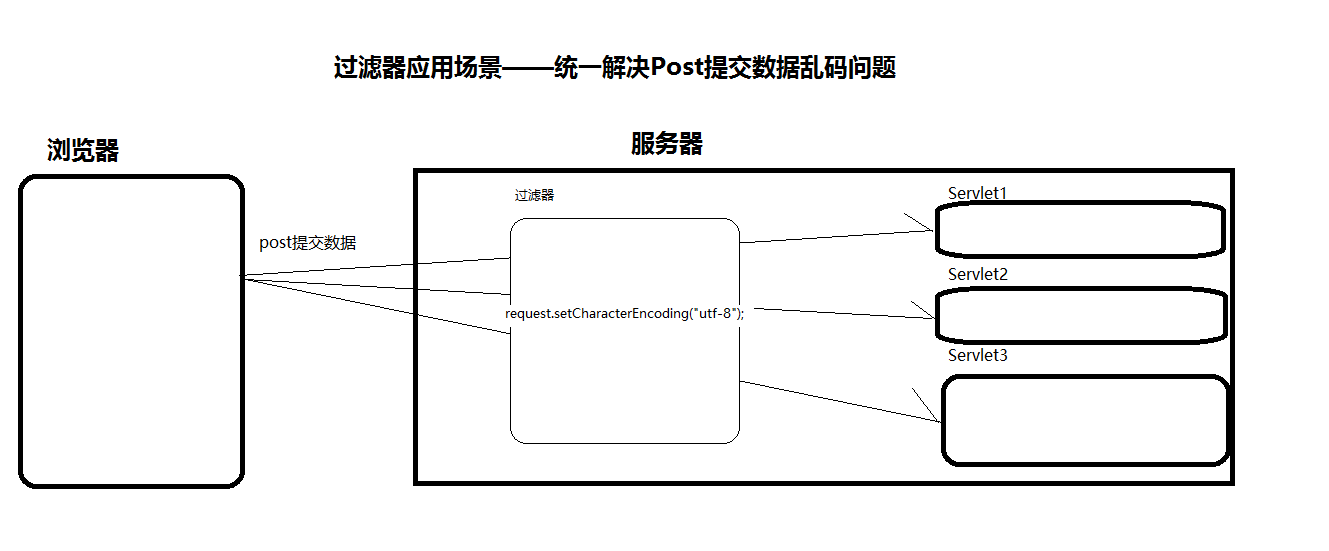
## 什么是过滤器

用来过滤用户的请求和响应，修改用户的请求和响应的数据，对请求进行拦截。

如果某个功能是很多Web资源都需要执行，我们就可以在过滤器中集中进行处理。

# 过滤器的应用场景

## 统一解决post提交请求乱码应用



## 过滤论坛中用户发表内容中的非法字符

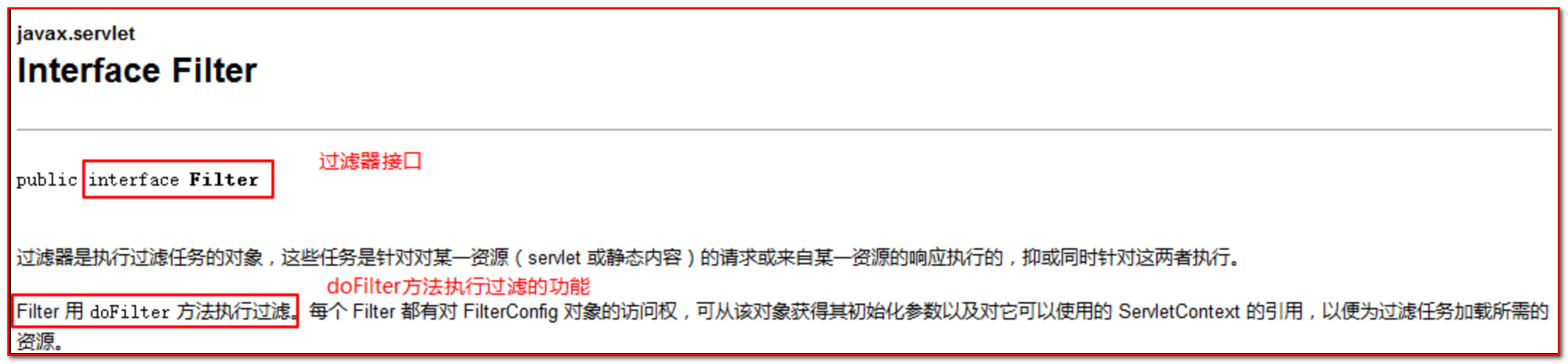
当我们上论坛或者贴吧这类的网站浏览信息时，我们发现，很多评论有时候是携带脏词的。而这些脏词不能显示出来，使用\*\*代替了。类似这类的脏词过滤，我们可以通过java中的过滤器，对请求参数中的脏词进行过滤。

## 登录权限检查

当客户端浏览器发出一个请求，这个请求在访问到正真的目标资源之前，我们需要进行登录权限的检查。如果已经登录，我们就让这个请求通过，访问资源；如果没有登录，我们不能让请求访问目标资源。这样的操作需要在访问具体资源进行条件的过滤，我们可以使用过滤器来完成。

# 我的第一个过滤器开发步骤

## Filter的API介绍



总结：

​ 1.我们创建一个过滤器的话需要实现Filter这个接口

​ 2.doFilter方法执行过滤器的功能

## 使用步骤

1. 创建一个类实现过滤器接口
2. 注解配置过滤器拦截的请求路径
3. 在doFilter方法中书写过滤任务
4. FilterChain.doFilter方法放行

## 注意事项

​ 过滤器doFilter方法默认拦截请求，如果需要经过过滤器之后，可以继续访问资源，要使用filterChain放行。

## 代码演示

过滤器代码

|  |
| --- |
| **package** com.itheima.filter;  **import** javax.servlet.\*; **import** javax.servlet.annotation.WebFilter; **import** java.io.IOException;  */\* \* urlPatterns = "/demo1Servlet" \* 是过滤器要过滤器的请求路径 \* \*/* @WebFilter(filterName = **"Demo1Filter"**,urlPatterns = **"/demo1Servlet"**) **public class** Demo1Filter **implements** Filter {   */\*\*  \* 进行过滤请求和响应的方法  \** ***@param request*** *\** ***@param response*** *\** ***@param chain*** *\** ***@throws*** *ServletException  \** ***@throws*** *IOException  \*/* **public void** doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) **throws** ServletException, IOException {  System.***out***.println(**"==正在执行Demo1Filter的请求过滤代码=="**);  chain.doFilter(request, response);*//放行，有这句代码就是放请求给资源去资源，如果没有请求不会到达资源* System.***out***.println(**"==正在执行Demo1Filter的响应过滤代码=="**); }     **public void** destroy() {  }     **public void** init(FilterConfig config) **throws** ServletException {   }  } |

Servlet代码​

|  |
| --- |
| **package** com.itheima.servlet;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"Demo1Servlet"**, urlPatterns = **"/demo1Servlet"**) **public class** Demo1Servlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  System.***out***.println(**"demo1Servlet正在执行。。。"**);  } } |

输出效果

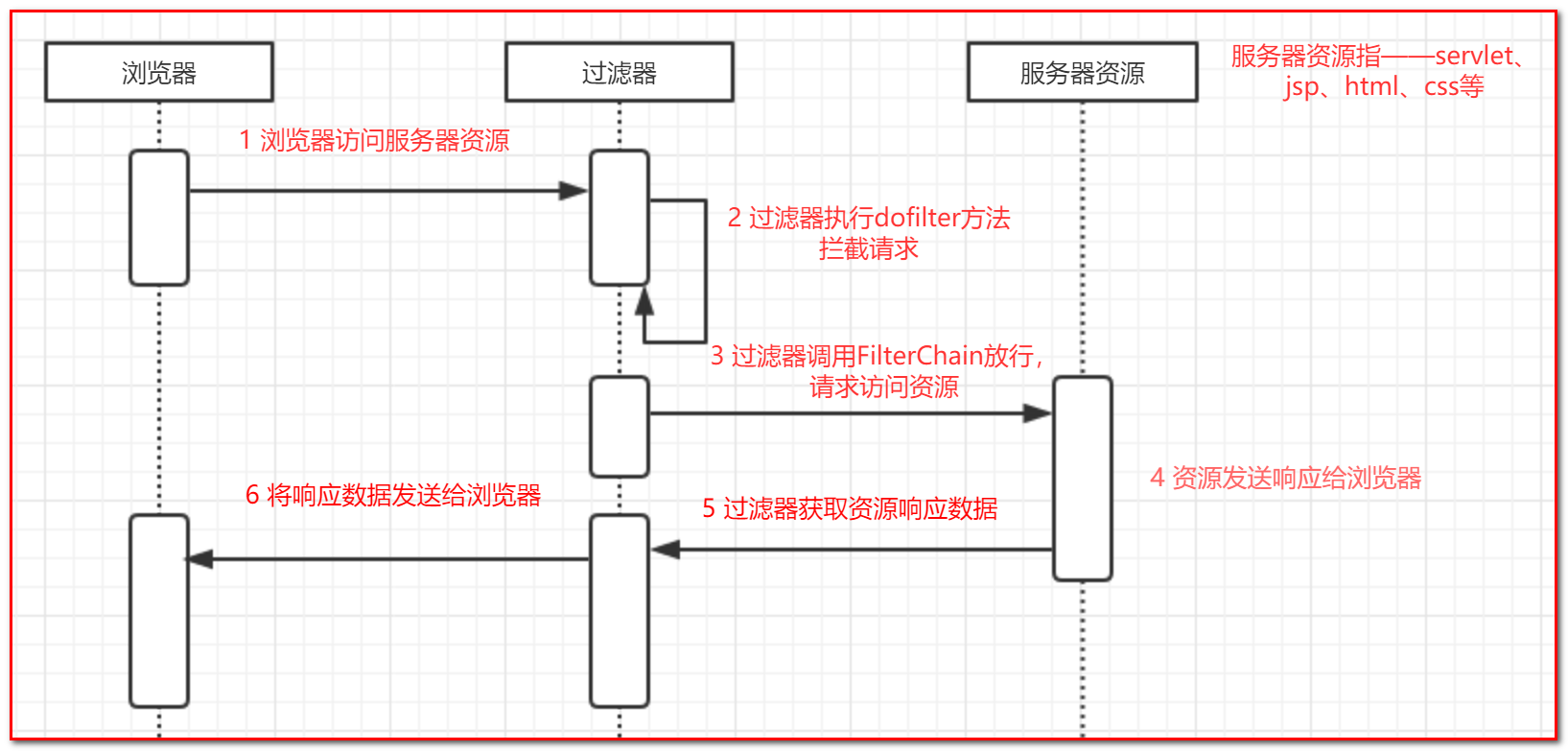
|  |
| --- |
| ==正在执行Demo1Filter的请求过滤代码==  demo1Servlet正在执行。。。  ==正在执行Demo1Filter的响应过滤代码== |

# 过滤器的执行流程

## 生活案例类比过滤器流程

璐璐从广州开车到长沙，中途途径韶关收费站。如果这座收费站业务通过了，韶关收费站放行，璐璐方可到达长沙。但是如果没有通过韶关收费站的业务，就不能到达长沙目的地。而璐璐中途经过的收费站，就相当于我们java中的过滤器。

## 图解过滤器的执行流程



# 过滤器的生命周期

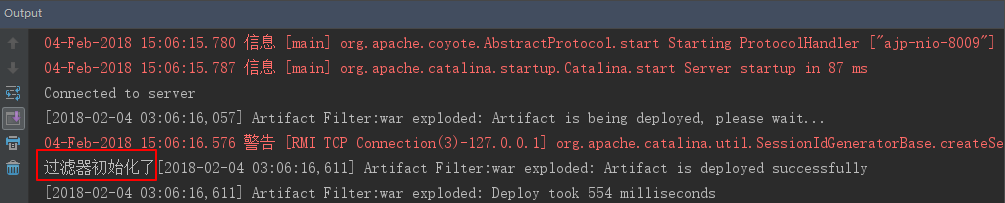
​ 过滤器我们已经看到了他的执行流程，但是，过滤器只有被创建出来才能执行，因此，我们还要了解过滤器生命周期。它包括了init、doFilter、destroy三个方法。

## init方法

API介绍

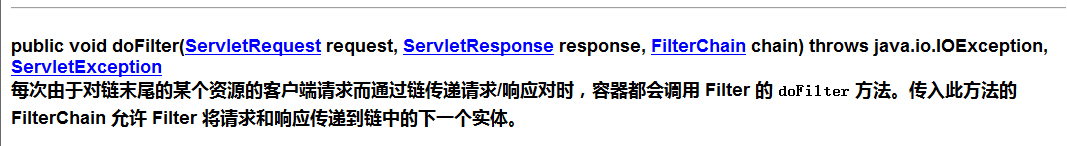


init方法是初始化方法。当过滤器被部署到web服务器时，服务器启动的时候执行一次。



## doFilter方法

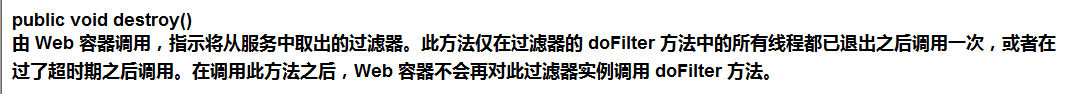
API介绍



只要一个请求符合Filter拦截路径，都会执行doFilter。需要注意的是:如果需要让这个请求通过过滤器，继续访问目标资源，一定得调用FilterChain对象的doFilter方法，要不然请求就被过滤器拦截了。FilterChain对象的doFilter方法表示对于请求放行。

## destory方法

AP介绍



过滤器的销毁方法。服务器停止或者将项目从服务器中移除的时候，销毁过滤器，执行一次destory方法。

## 代码

|  |
| --- |
| **package** com.itheima.filter;  **import** javax.servlet.\*; **import** javax.servlet.annotation.WebFilter; **import** java.io.IOException;  */\* \* urlPatterns = "/demo1Servlet" \* 是过滤器要过滤器的请求路径 \* \*/* @WebFilter(filterName = **"LifeCycleFilter"**,urlPatterns = **"/demo2Servlet"**) **public class** LifeCycleFilter **implements** Filter {   */\*\*  \* 初始化方法，过滤器实例的时候调用，服务器启动时创建  \** ***@param config*** *\** ***@throws*** *ServletException  \*/* **public void** init(FilterConfig config) **throws** ServletException {  System.***out***.println(**"==LifeCycleFilter的init方法调用了=="**);  }    */\*\*  \* 进行过滤请求和响应的方法,每次符合过滤请求路径的时候都会调用一次  \*  \** ***@param request*** *\** ***@param response*** *\** ***@param chain*** *\** ***@throws*** *ServletException  \** ***@throws*** *IOException  \*/* **public void** doFilter(ServletRequest request, ServletResponse response,  FilterChain chain) **throws** ServletException, IOException {  System.***out***.println(**"==正在执行LifeCycleFilter的请求过滤代码=="**);  chain.doFilter(request, response);*//放行，有这句代码就是放请求给资源去资源，如果没有请求不会到达资源* System.***out***.println(**"==正在执行LifeCycleFilter的响应过滤代码=="**); }    */\*\*  \* 销毁方法，服务器关闭或重启的时候调用  \*/* **public void** destroy() {  System.***out***.println(**"==LifeCycleFilter的destroy方法调用了=="**);  }     } |

## 运行结果

停止tomcat服务器：查看控制台

## 过滤器生命周期总结

1. 当服务器启动的时候，过滤器就被初始化了，执行过滤器的init方法
2. 每当一个请求的路径是满足过滤器的配置路径，那么就会执行一次过滤器的doFilter方法
3. 当服务器停止的时候，销毁过滤器，执行过滤器的destory方法

# 映射路径

假设有一个管理员权限的过滤器，它应该对用户发出的管理员功能的请求进行条件的过滤。但是当用户发出登录、注册等请求的时候，不应该进行过滤。所以我们过滤器，应该有选择的过滤器请求。这就需要学习配置过滤器不同的映射路径，从而让过滤器过滤希望过滤器的请求。

## 精确匹配模式

请求的路径需要与过滤器配置的路径完全一致，才能被这个过滤器过滤。如下4个案例

### /index.jsp

1.创建一个过滤器UrlFilter，配合过滤器的过滤路径为/index.jsp,过滤器代码如下：

package com.itheima.myfilter;  
​  
import javax.servlet.\*;  
import javax.servlet.annotation.WebFilter;  
import java.io.IOException;  
​  
@WebFilter(filterName = "UrlFilter",urlPatterns = "/index.jsp")  
public class UrlFillter implements Filter {  
    public void destroy() {  
   }  
​  
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException {  
        System.out.println("\*\*\*\*\*\*\*\*\*urlFilter拦截了请求\*\*\*\*\*\*\*\*\*\*\*");  
        chain.doFilter(req, resp);  
   }  
​  
    public void init(FilterConfig config) throws ServletException {  
​  
   }  
​  
}

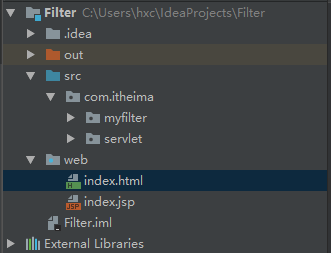
在浏览器输入路径<http://localhost:8080/day40/index.jsp> 的时候，才会经过当前过滤器。除了当前请求之外的请求都不会经过这个过滤器。因为过滤器配置的过滤路径/index.jsp只会对index.jsp这个首页的请求进行拦截过滤，其他请求不会拦截过滤。

### /index.html

1.修改UrlFilter过滤器的注解配置的过滤路径为/index.html,代码如下。

@WebFilter(filterName = "UrlFilter",urlPatterns = "/index.html")

2.web工程下创建一个index.html资源，资源位置和代码如下：



index.html代码：

<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title>Title</title>  
</head>  
<body>  
这是idnex.html页面  
</body>  
</html>

3.浏览器发出请求<http://localhost:9090/index.html>

查看控制台，发现当前请求会经过过滤器。除了访问index.html的请求外，其他请求都不会经过过滤器

### /DemoServlet

1.修改UrlFilter过滤器的注解配置的过滤路径为/DemoServlet,代码如下:

@WebFilter(filterName = "UrlFilter",urlPatterns = "/DemoServlet")

2.DemoServlet资源就使用5.2.3案例中的DemoServlet

3浏览器发出请求<http://localhost:9090/DemoServlet>

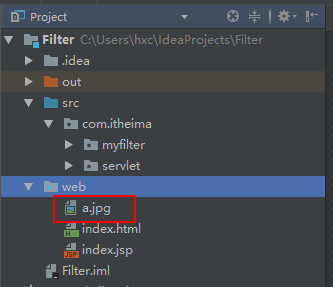
查看控制台，发现当前请求会经过过滤器。除了访问index.html的请求外，其他请求都不会经过过滤器

### /a.jpg

1.修改UrlFilter过滤器的注解配置的过滤路径为/a.jpg,代码如下:

@WebFilter(filterName = "UrlFilter",urlPatterns = "/a.jpg")

2.在项目中添加a.jpg资源，资源位置如下：



3浏览器发出请求<http://localhost:9090/a.jpg>

查看控制台，发现当前请求会经过过滤器。除了访问a.jpg的请求外，其他请求都不会经过过滤器

## 模糊匹配模式

通配符只有一个“\*”

### /\* 整个应用的配置

1.修改UrlFilter过滤器的注解配置的过滤路径为/\*,代码如下:

@WebFilter(filterName = "UrlFilter",urlPatterns = "/\*")

2.浏览器发出当前项目的任何请求都会经过当前过滤器

### /admin/\* 指定目录下的路径配置

1.修改UrlFilter过滤器的注解配置的过滤路径为/admin/\*,代码如下:

@WebFilter(filterName = "UrlFilter",urlPatterns = "/admin/\*")

2.浏览器发出当前项目下admin下的任何请求，都会经过过滤器。例如：

1.<http://localhost:8080/day40/admin/ss>

2.<http://localhost:8080/day40/admin/bb/aa>

# 拦截方式

有了上面学习的映射路径，我们可以控制过滤器过滤指定的内容，但是我们在访问资源的时候，并不是每次都是直接访问，有时是以转发的方式访问的，过滤器为了区分不同的访问资源的方式，有不同的拦截方式。

## request

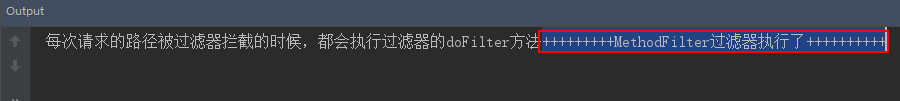
request是默认的拦截方式：浏览器发出的请求都会进行拦截。1.直接从地址栏访问index.jsp，Filter拦截2.重定向到index.jsp，Filter拦截

### 浏览器地址栏访问代码演示

1.创建过滤器MethodFilter,配置过滤路径为/index.jsp，过滤器代码如下：

package com.itheima.myfilter;  
​  
import javax.servlet.\*;  
import javax.servlet.annotation.WebFilter;  
import java.io.IOException;  
​  
@WebFilter(filterName = "MethodFilter",urlPatterns = "/index.jsp")  
public class MethodFilter implements Filter {  
    public void destroy() {  
   }  
​  
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException, IOException {  
        System.out.println("+++++++++MethodFilter过滤器执行了++++++++++");  
        chain.doFilter(req, resp);  
   }  
​  
    public void init(FilterConfig config) throws ServletException {  
​  
   }  
​  
}  
​

2.浏览器发出请求<http://localhost:8080/day40/index.jsp>，查看控制台：

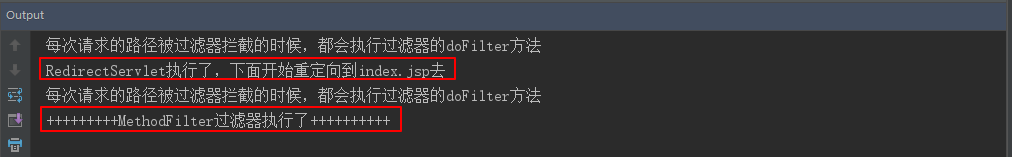


### 重定向访问代码演示

1.创建RedirectServlet,重定向到index.jsp去，代码如下：

package com.itheima.servlet;  
​  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
​  
@WebServlet(name = "RedirectServlet",urlPatterns = "/RedirectServlet")  
public class RedirectServlet extends HttpServlet {  
    protected vopackage com.itheima.servlet;  
​  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
​  
@WebServlet(name = "RedirectServlet",urlPatterns = "/RedirectServlet")  
public class RedirectServlet extends HttpServlet {  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        doGet(request,response);  
   }  
​  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        System.out.println("RedirectServlet执行了，下面开始重定向到index.jsp去");  
        response.sendRedirect(request.getContextPath()+"/index.jsp");  
   }  
}

2.浏览器地址栏输入地址<http://localhost:8080/day40/RedirectServlet>，控制台结果如下



可以发现，在执行重定向的时候，执行了过滤器

## forward

按照过滤器默认的拦截方式，我们只能拦截浏览器对服务器的之间访问，但是，如果是服务器资源之间使用转发的方式，就无法访问了，因此，我们还有一种forward拦截方式：当前拦截方式表示只有在进行请求的forward转发时，才会拦截请求进行过滤。只在转发到index.jsp时，Filter拦截

### forward转发代码演示

1.创建ForwardServlet，转发到index.jsp去，代码如下

package com.itheima.servlet;  
​  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
​  
@WebServlet(name = "ForwardServlet",urlPatterns = "/ForwardServlet")  
public class ForwardServlet extends HttpServlet {  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        doGet(request,response);  
   }  
​  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        System.out.println("ForwardServlet执行了，下面开始转发请求到index.jsp去");  
        request.getRequestDispatcher("/index.jsp").forward(request,response);  
   }  
}  
​

2.浏览器地址栏输入地址：<http://localhost:8080/day40/ForwardServlet>，查看控制台，发现请求转发到index.jsp,访问了index.jsp资源，但是并没有被MethodFilter过滤器拦截。这是因为默认的request拦截方式对于转发的请求不会进行拦截。

3.修改MethodFilter的拦截方式为forward。修改MethodFilter注解代码，添加代码dispatcherTypes = DispatcherType.FORWARD

注解代码如下：

@WebFilter(filterName = "MethodFilter",urlPatterns = "/index.jsp",dispatcherTypes = DispatcherType.FORWARD)

4.重新启动服务器，浏览器再次访问请求：<http://localhost:8080/day40/ForwardServlet>，查看控制台，发现当请求转发到index.jsp的时候，会经过过滤器的过滤。

## 两种拦截方式能否共存

那么有了两种拦截方式，两种拦截方式能否共存呢？答案是：可以的。

|  |
| --- |
| **package** com.itheima.filter;  **import** javax.servlet.\*; **import** javax.servlet.annotation.WebFilter; **import** java.io.IOException;  */\* \* \* 过滤器拦截方式  \* request（默认方式）,过滤直接访问的请求路径（浏览器直接访问和后端重定向）  \* forward，过滤页面跳转请求转发的请求路径  \*  \* dispatcherTypes = {DispatcherType.REQUEST} 配置拦截方式，request是默认的。如果没有配置拦截方式默认是request，如果配置了就只是配置的拦截方式  \* dispatcherTypes = {DispatcherType.FORWARD} 如果拦截方式只配置了一种，那么request方式就无效  \* \* \*/* @WebFilter(filterName = **"DispatcherFilter"**,urlPatterns = **"/Demo4Servlet"**,dispatcherTypes = {DispatcherType.***FORWARD***,DispatcherType.***REQUEST***}) **public class** DispatcherFilter **implements** Filter {  **public void** destroy() {  }   **public void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain) **throws** ServletException, IOException {  System.***out***.println(**"==正在执行DispatcherFilter的请求过滤代码=="**);  chain.doFilter(request, response);*//放行，有这句代码就是放请求给资源去资源，如果没有请求不会到达资源* System.***out***.println(**"==正在执行DispatcherFilter的响应过滤代码=="**);  }   **public void** init(FilterConfig config) **throws** ServletException {   }  } |

配置成如上，我们会发现

直接访问Demo4Servlet和Demo3Servlet转发到Demo4Servlet都会执行过滤器

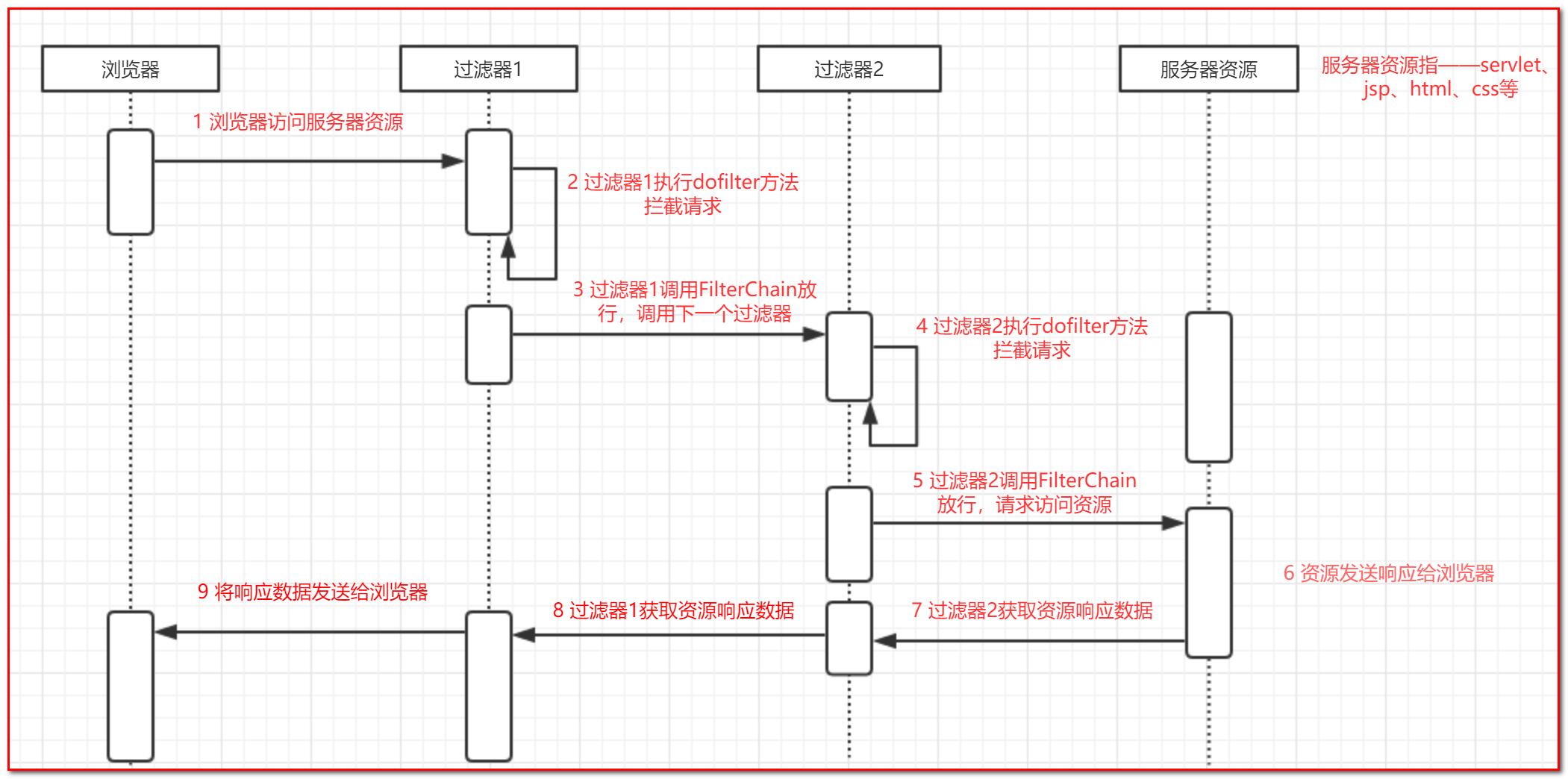
# 过滤器链

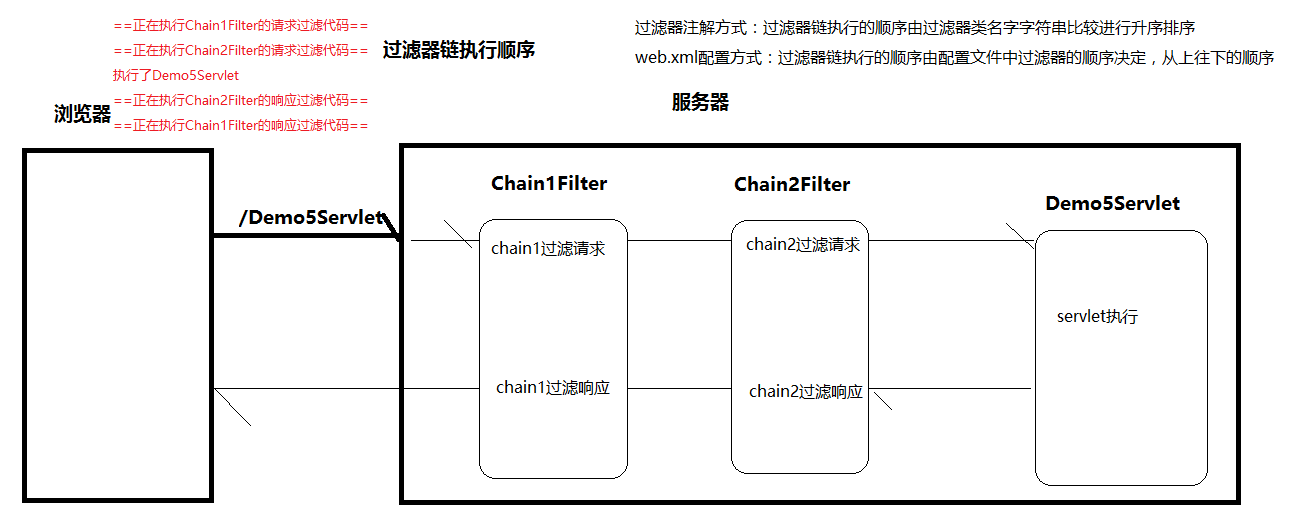
## 什么是过滤器链

璐璐从广州开车到长沙，途径韶关收费站，株洲收费站。韶关收费站收的是路过韶关的费用，株洲收费站收的是路过株洲的费用。如果这二座收费站业务通过了，方可到达长沙，只要其中一个收费站的业务没有通过，那么就不能到达长沙。而且，只有通过了韶关收费站，才能到达株洲收费站。

同样，我们java代码中，有时需要对同一个请求，进行多次不同业务的过滤，所以我们java代码中需要多个过滤器。只有所有的过滤器都对请求进行了放行，请求才能访问到目标资源，只要多个过滤器中只有一个过滤器不放行请求，那么这个请求都不能够访问到目标资源。多个过滤器组成的一个整体我们称为过滤器链。而且，过滤器链中的过滤器是一个一个的执行的，一个过滤器执行完毕之后，会执行下一个过滤器，后面没有过滤器了，才会访问到目标资源。只要其中一个过滤器没有放行，那么这个过滤器后面的过滤器也都不会执行了。

## 过滤器链的执行流程





创建Chian1Filter

|  |
| --- |
| **package** com.itheima.filter;  **import** javax.servlet.\*; **import** javax.servlet.annotation.WebFilter; **import** java.io.IOException;  */\*@WebFilter(filterName = "Chain1Filter",urlPatterns = "/Demo5Servlet")\*/* **public class** Chain1Filter **implements** Filter {  **public void** destroy() {  }   **public void** doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) **throws** ServletException, IOException {  System.***out***.println(**"==正在执行Chain1Filter的请求过滤代码=="**);  chain.doFilter(req, resp);*//放行，有这句代码就是放请求给资源去资源，如果没有请求不会到达资源* System.***out***.println(**"==正在执行Chain1Filter的响应过滤代码=="**);  }   **public void** init(FilterConfig config) **throws** ServletException {   }  } |

创建Chian2Filter

|  |
| --- |
| **package** com.itheima.filter;  **import** javax.servlet.\*; **import** javax.servlet.annotation.WebFilter; **import** java.io.IOException;  */\*@WebFilter(filterName = "Chain2Filter",urlPatterns = "/Demo5Servlet")\*/* **public class** Chain2Filter **implements** Filter {  **public void** destroy() {  }   **public void** doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) **throws** ServletException, IOException {  System.***out***.println(**"==正在执行Chain2Filter的请求过滤代码=="**);  chain.doFilter(req, resp);*//放行，有这句代码就是放请求给资源去资源，如果没有请求不会到达资源* System.***out***.println(**"==正在执行Chain2Filter的响应过滤代码=="**);  }   **public void** init(FilterConfig config) **throws** ServletException {   }  } |

Demo5Servlet

|  |
| --- |
| **package** com.itheima.servlet;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"Demo5Servlet"**, urlPatterns = **"/Demo5Servlet"**) **public class** Demo5Servlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  System.***out***.println(**"执行了Demo5Servlet"**);  } } |

配置web.xml(使用这种方式，过滤器链执行属性由在配置文件中配置从上到下顺序执行)

|  |
| --- |
| *<!--配置过滤器-->* <**filter**>  <**filter-name**>Chain2Filter</**filter-name**>  <**filter-class**>com.itheima.filter.Chain2Filter</**filter-class**> </**filter**> <**filter-mapping**>  <**filter-name**>Chain2Filter</**filter-name**>  <**url-pattern**>/Demo5Servlet</**url-pattern**> </**filter-mapping**>  <**filter**>  <**filter-name**>Chain1Filter</**filter-name**>  <**filter-class**>com.itheima.filter.Chain1Filter</**filter-class**> </**filter**> <**filter-mapping**>  <**filter-name**>Chain1Filter</**filter-name**>  <**url-pattern**>/Demo5Servlet</**url-pattern**> </**filter-mapping**> |

总结



# 案例：非法字符

## 案例需求

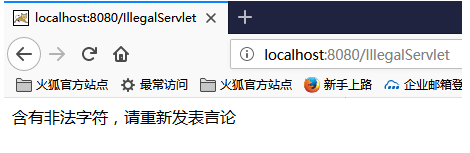
当用户发出非法言论的时候，提示用户言论非法。

## 案例效果

1.在表单中输入含有非法字符的言论，点击提交按钮



2.控制台显示如下：



## 案例分析

1.创建一个表单用于发表言论。

2.创建一个txt文件，其中存入非法字符。

3.创建一个Filter，拦截请求。在init方法中将txt文件中的非法字符读取到内存中。

4.获取请求中的参数，对请求的参数进行非法字符的校验。

5.如果言论中不含有非法字符，就放行。

6.如果言论中含有非法字符，就拦截，并且提示用户非法言论。

## 实现步骤

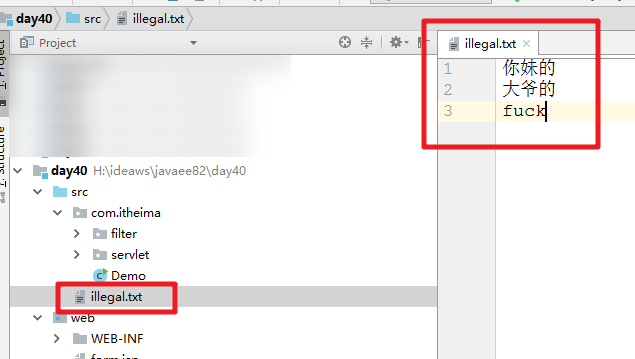
1.创建一个表单，用于发表言论

form1.jsp代码如下：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
    <form method="post" action="IllegalServlet">  
       请发表你的言论：<input type="text" name="message">  
        <input type="submit" value="提交">  
    </form>  
</body>  
</html>

2.创建一个txt文件，存入非法字符

例如：



3.创建一个servlet用于接受表单提交的内容

IllegalServlet代码如下：

package com.itheima.servlet;  
​  
import javax.servlet.ServletException;  
import javax.servlet.annotation.WebServlet;  
import javax.servlet.http.HttpServlet;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
​  
@WebServlet(name = "/IllegalServlet",urlPatterns = "/IllegalServlet")  
public class IllegalServlet extends HttpServlet {  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        doGet(request,response);  
   }  
​  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        response.getWriter().write("你的言论发表成功");  
   }  
}

4.创建一个过滤器，用来拦截请求，过滤请求中发表的言论的非法字符

IllegalFilter代码如下：

|  |
| --- |
| **package** com.itheima.filter;  **import** javax.servlet.\*; **import** javax.servlet.annotation.WebFilter; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.\*; **import** java.util.ArrayList; **import** java.util.List;  @WebFilter(filterName = **"IllegalFilter"**,urlPatterns = **"/illegal"**) **public class** IllegalFilter **implements** Filter {  **public void** destroy() {  }     **public void** doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) **throws** ServletException, IOException {  *//将父接口转换为子接口* HttpServletRequest request = (HttpServletRequest) req;  HttpServletResponse response = (HttpServletResponse) resp;   *//过滤发表言论请求，将言论里面非法词汇拒绝发表   //处理post提交请求乱码* request.setCharacterEncoding(**"utf-8"**);  response.setContentType(**"text/html;charset=utf-8"**);  *//1.获取用户的请求数据* String message = request.getParameter(**"message"**);   *//获取非法词汇列表（我们认为的）对比用户输入的数据* **for**(String word :**illegalList**){  **if**(message.indexOf(word)>-1){  response.getWriter().write(**"您的言论含有敏感词汇，请求重新发表引论"**);  *//response.getWriter().write("<script>history.back()</script>");* **return** ;*//停止当前处理* }  }   chain.doFilter(req, resp);  }    *//存储到一个集合中* **private** List<String> **illegalList** = **new** ArrayList<String>();  **public void** init(FilterConfig config) **throws** ServletException {  *//读取非法词汇文件内容存储内存对象* **try** {  *//1.获取illegal.txt的绝对路径* String path = config.getServletContext().getRealPath(**"/WEB-INF/classes/illegal.txt"**);  *//2.创建输入流* InputStreamReader inputStreamReader =  **new** InputStreamReader(**new** FileInputStream(path),**"utf-8"**);  *//3.缓存字符读取流* BufferedReader bufferedReader = **new** BufferedReader(inputStreamReader);   *//4.定义读取的数据行* String line = **null**;  **while** ((line=bufferedReader.readLine())!=**null**){  **illegalList**.add(line);  }  bufferedReader.close();*//由于inputStreamReader是被bufferedReader引用，所以关闭了bufferedReader自动会将inputStreamReader关闭  //inputStreamReader.close();* } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

# 综合案例：解决全站乱码

## 案例需求

浏览器发出的任何请求，通过过滤器统一处理中文乱码。

## 案例效果

分别以get方式和post方式提交中文，servlet中不做中文乱码处理，直接获取参数，得到的参数不存在中文乱码问题。

## 案例分析

1.创建一个过滤器。

2.因为对所有的请求进行乱码的过滤，所以过滤器的过滤路径配置为/\*

3.针对post请求处理乱码

## 实现步骤

1.创建一个form2.jsp表单，用于测试过滤器解决乱码的功能

2.form2.html代码：

<%@ page contentType="text/html;charset=UTF-8" language="java" %>  
<html>  
<head>  
    <title>Title</title>  
</head>  
<body>  
    <form action="encoding" method="post">  
        <input type="text" name="name"/>  
        <input type="submit" value="提交"/>  
    </form>  
</body>  
</html>

3.创建一个用于接收表单请求的EncodingServlet.

EncodingServlet代码：

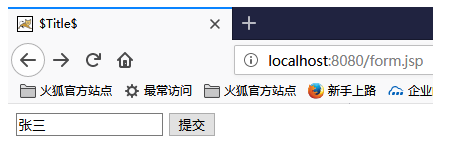
|  |
| --- |
| **package** com.itheima.servlet;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"EncodingServlet"**, urlPatterns = **"/encoding"**) **public class** EncodingServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  *//获取数据* String name = request.getParameter(**"name"**);  System.***out***.println(name);  } } |

4.创建EncodingFilter用于乱码处理的过滤器

EncodingFilter代码如下：

|  |
| --- |
| **package** com.itheima.filter;  **import** javax.servlet.\*; **import** javax.servlet.annotation.WebFilter; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebFilter(filterName = **"EncodingFilter"**,urlPatterns = **"/\*"**) **public class** EncodingFilter **implements** Filter {  **public void** destroy() {  }   **public void** doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) **throws** ServletException, IOException {  *//1.将父接口转换为子接口* HttpServletRequest request = (HttpServletRequest) req;  HttpServletResponse response = (HttpServletResponse) resp;  *//2.处理post提交请求乱码* String method = request.getMethod();  **if**(method.equalsIgnoreCase(**"post"**)){  *//处理post乱码* request.setCharacterEncoding(**"utf-8"**);  }  response.setContentType(**"text/html;charset=utf-8"**);   *//解决全站请求与输出乱码问题* chain.doFilter(request,response);  }   **public void** init(FilterConfig config) **throws** ServletException {   }  } |

5.浏览器访问form.jsp页面，表单提交方式为post，表单输入中文



6.点击提交按钮，查看控制台，post乱码问题得到解决



7.将表单的提交方式换成get，提交表单，查看控制台没有乱码问题。