Filter 过滤器

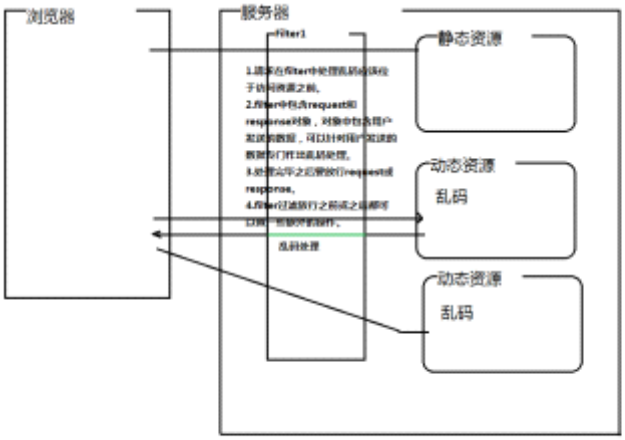
2019年5月25日 星期六 9:20

1. 过滤器概述

Servlet三大技术之一。实用性最高的技术之一。它能够实现如下功能：
作用：乱码过滤、响应压缩数据、网盘快传、权限设置、敏感词汇过滤

过滤器中存在责任链模式。

过滤器主要拦截的是访问服务器资源的请求和服务器响应的内容，拦截的数据可以进行处理，处理后可以放行或者不放行，若放行，放行前后都可以添加额外的代码。



2. 过滤器实现

a. 创建一个普通类，使类实现一个filter接口即可。
b. 配置过滤拦截请求的方式。

3. 过滤器的API

Method Summary	
void	destroy() Called by the web container to indicate to a filter that it is being taken out of service. 在当前web应用被移出容器（web被销毁的时候）调用次方法完成善后的操作。
void	doFilter(ServletRequest request, ServletResponse response, FilterChain chain) The doFilter method of the Filter is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.

	在每一次的请求或响应过程当中，都会访问一次filter中的doFilter方法，在这里可以对request和response进行处理，处理过后可以选择放行或者不放行，如果放行则在放行前后的代码中可以进行一些额外的操作。
void	init(FilterConfig filterConfig) Called by the web container to indicate to a filter that it is being placed into service. 在web应用创建的时候自动创建一个filter对象在当前tomcat内存当中，提供服务，创建对象的时候会自动调用一次init方法。

4. 过滤器实现

```
package cn.tedu.filter;

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;

public class FirstFilter implements Filter {

	public void init(FilterConfig filterConfig) throws ServletException {
		System.out.println("FirstFilter Start.....");
	}

	public void doFilter(ServletRequest request, ServletResponse response,
		FilterChain chain) throws IOException, ServletException {
		System.out.println("FirstFilter doFilter方法已经被访问");
		//放行当前请求或响应。
		chain.doFilter(request, response);
		System.out.println("FirstFilter doFilter方法已经被访问2");
	}

	public void destroy() {
		System.out.println("FirstFilter End.....");
	}

}
```

配置filter filter-mapping：

```
<filter>
	<filter-name>FirstFilter</filter-name>
	<filter-class>cn.tedu.filter.FirstFilter</filter-class>
 </filter>
 <filter-mapping>
	<filter-name>FirstFilter</filter-name>
	<url-pattern>/*</url-pattern>
```

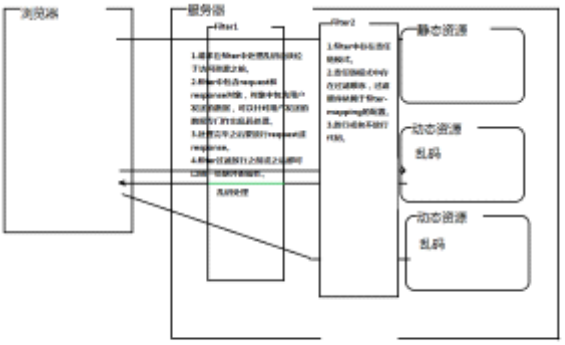
```
</filter-mapping>
```

5. Filter的生命周期

在web应用启动的时候filter会自动加载，并且创建一个对象驻留在内容当中，创建对象的时候会主动调用一次init方法完成初始化的操作。驻留在内存中的对象会一直拦截后续指定的请求，内存中只有第一次初始化的时候创建的唯一一个filter对象。在一个请求或响应被拦截的时候会调用filter身上的doFilter方法对请求或响应拦截，然后进行处理，处理后的结果可以放行，也可以不放行。如果放行，在放行前后可以添加额外的操作。在web容器被销毁的时候filter对象也会被销毁，在filter对象销毁之前会调用destory方法完成一些善后的操作。

6. 责任链模式

在过滤使用过程中，可以配置多个过滤器，过滤器按照web.xml文件中filter-mapping配置的顺序进行拦截，多个过滤器连接在一起就构成了一个责任链模式的过滤器。



doFilter方法执行前后代码都可以正常运行，在所有filter中的doFilter之前的代码会按照filter-mapping的配置顺序执行，在访问结束的时候，filter中doFilter之后的代码会按照filter-mapping配置相反的顺序执行。

修改EasyMall-全站乱码处理

2019年5月25日 星期六 10:34

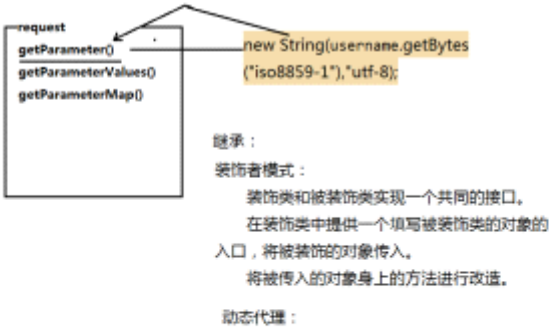
1. 需求

在www.easymall.com这个网站中，多个servlet都需要进行request和response乱码的处理，重复书写加大代码冗余，而且如果servlet众多，书写繁琐，可以将处理乱码的过程专门提取到一个模块中处理，将处理的请求和响应再发送到对应的资源或浏览器中，那么这个模块可以使用过滤器实现。

2. 过滤器实现全站乱码处理

由于request可以是post或者get提交，post提交通过一句request.setCharacterEncoding("utf-8")就可以解决，get提交需要将每一个请求参数单独做乱码处理，如果每个都处理十分繁琐，可以将这个过程提取到过滤器当中。

response乱码处理也可以放在filter中，这样所有的响应资源就都可以处理乱码了。



代码实现：

i. 创建一个Encodingfilter过滤器，并配置相关配置信息在web.xml中

```
package com.easymall.filter;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

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;
import javax.servlet.http.HttpServletRequestWrapper;
//全站乱码处理
public class EncodingFilter implements Filter {
	String encode = "";
	public void init(FilterConfig filterConfig) throws ServletException {
		encode = filterConfig.getServletContext().getInitParameter("encode");
```

```
    }

    class MyHttpServletRequest extends HttpServletRequestWrapper{

        public HttpServletRequest request = null;
//                                                          此处为用户传入的request对象
    public MyHttpServletRequest(HttpServletRequest request) {
        super(request);
        //将用户传入的request对象提取成一个成员变量，以便在其他方法中使用。
        this.request = request;
    }

    @Override
    public Map<String,String[]> getParameterMap() {
        try {
            //将处理后的结果数据放入一个新的map，并返回。
            //从旧map取出包含乱码的数据
            Map<String,String[]> map = request.getParameterMap();
            //新map， 存储处理后的没有乱码的数据
            Map<String,String[]> rmap = new HashMap<String,String[]>();
            //处理乱码数据
            for(Map.Entry<String, String[]> entry:map.entrySet()){
                //取出键和值，对其中的值进行乱码的处理
                String key = entry.getKey();
                String[] values = entry.getValue();
                //新的String数组，用于存储处理后的值
                String[] rvalues = new String[values.length];
                //对其中的值进行乱码的处理
                for(int i=0;i<values.length;i++){
                    rvalues[i] = new String(values[i].getBytes("iso8859-1"),encode);
                }
                //将处理后的结果放入新的map中
                rmap.put(key, rvalues);
            }
            //返回添加好正常中文的map对象
            return rmap;
        } catch (Exception e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }

    }
    @Override
    public String[] getParameterValues(String name) {
        //调用的是在本类中修改好的方法，千万不要加request，否则会变为原有的request对象
        身上的方法
        Map<String,String[]> map = getParameterMap();
        return map.get(name);
    }
    @Override
    public String getParameter(String name) {
        //根据用户传入的参数名获取对应名称的参数值，
        //结果为一个数组，取出数组中第一个元素即为getParameter方法的返回值
        String[] values = getParameterValues(name);
        //考虑方法中的参数可能不存在，
        //导致执行方法出现varlues=null的结果。
```

```
            //为了避免null[0]这种情况，应该对values结果做出如下判断。
            return values==null?null:values[0];


        }


        }
        public void doFilter(ServletRequest request, ServletResponse response,
                FilterChain chain) throws IOException, ServletException {
            //1.response对象
            response.setContentType("text/html;charset="+encode);
            //2.request对象 post
//                request.setCharacterEncoding("utf-8");
        //2.request对象 get
        MyHttpServletRequest req = new MyHttpServletRequest((HttpServletRequest) request);

        //放行 处理后的request对象
        chain.doFilter(req, response);

        }

        public void destroy() {

        }

}
```

ii. 在web.xml配置filter

```
<!-- 全站乱码过滤的filter -->
 <filter>
        <filter-name>EncodingFilter</filter-name>
        <filter-class>com.easymall.filter.EncodingFilter</filter-class>
 </filter>
 <filter-mapping>
        <filter-name>EncodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
 </filter-mapping>
```

iii. 由于在开发中可能会使用utf-8及其以外的字符集，在response对象身上设置字符集是不能写死，应该通过配置文件读取全局指定字符集。

```
public class EncodingFilter implements  Filter {
        String encode = "";
        public void init(FilterConfig filterConfig) throws  ServletException {
                encode = filterConfig.getServletContext().getInitParameter("encode");
        }
```

在web. xml中配置：

```
<!-- 全局字符集设置 -->
 <context-param>
        <param-name>encode</param-name>
        <param-value>utf-8</param-value>
 </context-param>
```