Filter 过滤器

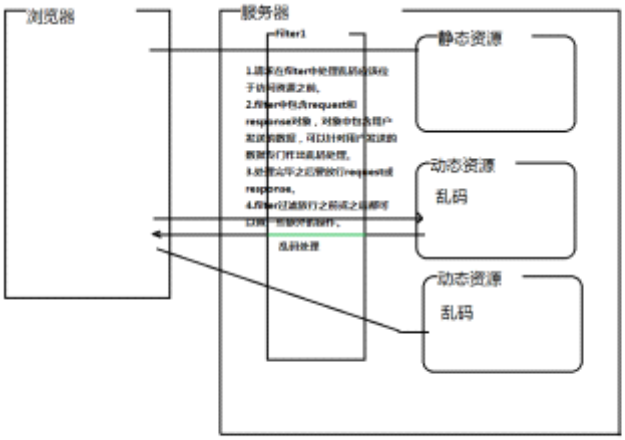
2019年5月25日 星期六 9:20

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>
```

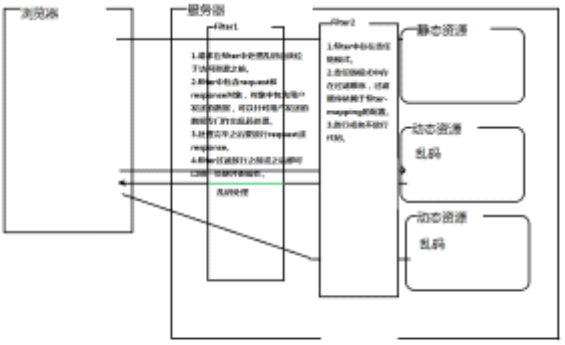
```
</filter-mapping>
```

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

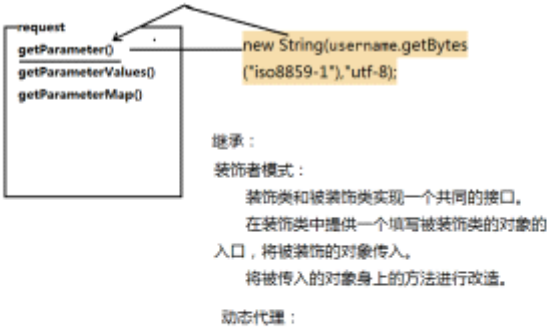
1. 需求

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

```
//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(encode);
```

```
        //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>
```

3. 拆分MyHttpServletRequest和EncodingFilter

EncodingFilter:

```
//全站乱码处理
public class EncodingFilter implements Filter {
        String encode = "";
        boolean use_encode = false;
        public void init(FilterConfig filterConfig) throws ServletException {
                encode = filterConfig.getServletContext().getInitParameter("encode");
                //是否开启字符集处理
                use_encode = Boolean.parseBoolean(filterConfig.getServletContext().getInitParameter("use_encode"));
        }


        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(encode);
        //2.request对象 get
        //在返回request对象的时候判断是否使用乱码处理的后的request对象
```

```
        ServletRequest req = use_encode? new MyHttpServletRequest((HttpServletRequest)  request,encode):request;

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

        }

        public void destroy() {

        }

}
```

MyHttpServletRequest:

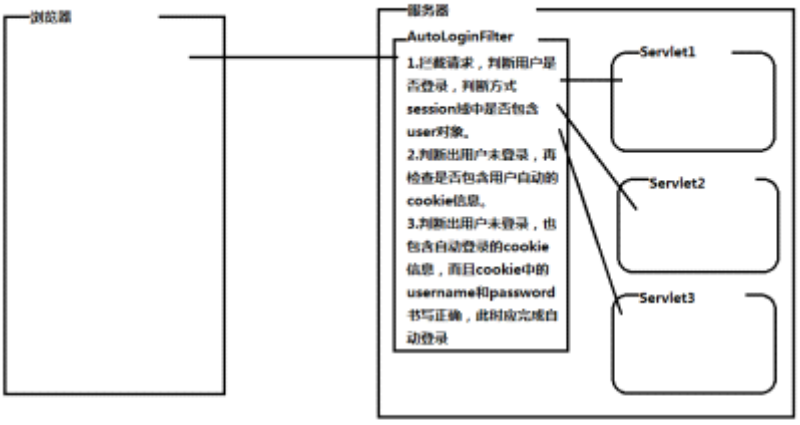
在使用get提交乱码处理的过程中，我们始终认为服务器一定为iso8859-1，但是实际上有些服务器可能是utf-8，有些则可能是iso8859-1，面对不同的服务器配置时，我们的代码应该实现一个通用的效果，所以为了实现这个效果，我们需要添加一个控制是否使用乱码处理的开关，开关打开则启用乱码处理，开关关闭则取消乱码处理。

```
public class MyHttpServletRequest extends HttpServletRequestWrapper{

        public HttpServletRequest request = null;
        public String encode = "";
//                                                              此处为用户传入的request对象
        public MyHttpServletRequest(HttpServletRequest  request,String encode) {
                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;
                this.encode = encode;
        }

        @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];


        }
}
```

4. EasyMall自动登录

在login. jsp页面中有一个复选框--30天自动登录，选中之后应该在每一次打开浏览器访问EasyMall服务器上任意一个资源的时候都能实现自动登录的效果。为了实现这个效果，结果如下图分析，决定使用filter和cookie两项技术解决这个问题。



代码实现：

i. 创建AutoLoginFilter：

```
//30天内自动登录
public class AutoLoginFilter implements Filter {

    public UserService userService = new UserService();
    public void init(FilterConfig filterConfig) throws ServletException {

    }

    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        //1.判断是否登录 根据session域中是否包含user来判断用户是否登录
        HttpServletRequest req = (HttpServletRequest) request;
        //判断session对象是否存在，如果session不存在，则没有登录状态，
```

```
            //如果有session而没有user域属性，
            //也可以认为是没有登录状态
            if(req.getSession(false) == null
                    ||req.getSession().getAttribute("user")==null){
                    //没有登录状态
                    Cookie[] cs = req.getCookies();
                    Cookie autoLoginC = null;
                    if(cs != null){
                            for(Cookie c : cs){
                                    if("autologin".equals(c.getName())){
                                            autoLoginC = c;
                                    }
                            }

                    }
                    //autologinC有可能未找到，所以需要先判断是否为空
                    if(autoLoginC != null){
                            //获取cookie中的用户名和密码
                            String str = autoLoginC.getValue();
                            String[] values = str.split("#");
                            String username = URLDecoder.decode(values[0], "utf-8");
                            String password = values[1];
                            //完成自动登录
                            //根据数据库信息查询用户名和密码事如否匹配
                            User user = userService.loginUser(username, password);
                            req.getSession().setAttribute("user", user);
                    }

            }

            //不论登录用户名和密码是否正确都应该放行请求和响应。
            chain.doFilter(request, response);

    }

    public void destroy() {

    }

}
```

ii. 配置web.xml，添加filter配置信息

```
<!-- 30天内自动登录 -->
 <filter>
        <filter-name>AutoLoginFilter</filter-name>
        <filter-class>com.easymall.filter.AutoLoginFilter</filter-class>
 </filter>
 <filter-mapping>
        <filter-name>AutoLoginFilter</filter-name>
        <url-pattern>/*</url-pattern>
 </filter-mapping>
```

5. LogOutServlet代码修改：

原有的注销功能只是删除session，访问原有的注销功能仍然会发生一次请求，这个请求还会被AutoLogin拦截，并且再次自动登录，导致无法注销，所以应该在其中添加删除autologinCookie的代码

```
//注销用户登录状态
public class LogOutServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
            //注销用户登录状态可以将session删除，删除session后其中的属性值也会销毁
            //实现用户注销的功能
            //先判断session对象是否为空，如果为空则不需要释放。
```

```
            //不为空再释放。
            if(request.getSession(false)!=null){
                request.getSession().invalidate();
            }
            //30天内自动登录清除cookie，避免点击之后重复登录，需要等待30天
            Cookie cookie = new Cookie("autologin","");
            cookie.setMaxAge(0);
            cookie.setPath(request.getContextPath()+"/");
            response.addCookie(cookie);

            response.sendRedirect("http://www.easymall.com");
        }

        public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            doGet(request, response);

        }

}
```

MD5加密算法

2019年5月25日 星期六 15:54

1. MD5加密算法

   又称作数据摘要算法。MD5可以对重要数据进行加密，还可以对重要文件加密，加密之后文件会产生一个加密结果，所有MD5加密后都应该得到这个结果，如果结果发生变化则证明当前文件已经被其他人修改。利用这个特点可以校验文件的完整性和正确性。

   md5作用：

   秒传

   加密

   文件完整性和正确性

   王小云 MD5破解网站

2. MD5加密算法的引入

   课前资料中的MD5加密.txt文件，全部内容复制到一个MD5Utils中

3. 利用MD5修改EasyMall

   a. 加密用户密码：

      i. 修改servlet中的password

         1) loginServlet：

```
//判断30天内是否自动登录
if("true".equals(autologin)){
    //将cookie存储30天
    Cookie cookie = new
    Cookie("autologin",URLEncoder.encode(username,"utf-8")+"#"+MD5Utils.md5(passwo
    rd));
    cookie.setMaxAge(60*60*24*30);
    cookie.setPath(request.getContextPath()+"/");
    response.addCookie(cookie);
}

//UserService负责接收javabean提供的数据
    //UserService提供处理数据的方法
    //将登陆功能的返回结果作为保存用户登陆信息使用。所以需要返回值对象
    User user =null;
    try{
        user = userService.loginUser(username,MD5Utils.md5(password));

    }catch (MsgException e) {
        request.setAttribute("msg", e.getMessage());
        request.getRequestDispatcher("/login.jsp").forward(request, response);
        return;
    }
    //将用户信息保存仅入session域
```

         2) registServlet：

```
//UserService
    User user = new User(0,username,MD5Utils.md5(password),nickname,email);
    try{
        userService.registUser(user);
```

```
} catch (MsgException e) {
        request.setAttribute("msg",e.getMessage());
        request.getRequestDispatcher("/regis.jsp").forward(request, response);
        return;
}
```

ii. 修改数据库中的密码为MD5加密算法的结果

```
mysql> update user set password = md5(password);
Query OK, 12 rows affected (0.06 sec)
Rows matched: 12  Changed: 12  Warnings: 0
```

```
update user set password = md5(password);
```

Listener监听器

2019年5月25日 星期六 16:32

1. listener概述

servlet三大技术之一。用于监听某一个域或者域中属性的变化，一旦数据发生改变，则会触发监听器执行对应的操作。

2. Listener监听的分类

JAVAEE开发中，监听器共有三类八种。

a. 监听三大作用域创建和销毁的监听器

```
ServletContextListener
HttpSessionListener
ServletRequestListener
```

代码实现：

i. 创建监听器的类并实现接口：

```
package cn.tedu.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
//ServletContext对象的监听器
public class SCListener implements ServletContextListener {

	//对象创建时候触发的方法
	public void contextInitialized(ServletContextEvent sce) {
		System.out.println("ServletContext对象已经创建"+sce.getServletContext());

	}
	//对象销毁的时候触发的方法
	public void contextDestroyed(ServletContextEvent sce) {
		System.out.println("ServletConxt对象已经销毁"+sce.getServletContext());

	}

}
```

ii. 在web.xml文件中配置listener

```
<!-- 配置监听器-->
 <listener>
	<listener-class>cn.tedu.listener.SCListener</listener-class>
 </listener>
```

b. 监听三大作用域中属性变化的监听器

```
ServletContextAttributeListener
HttpSessionAttributeListener
ServletRequestAttributeListener
```

i. 代码实现：

```
package cn.tedu.listener;

import javax.servlet.ServletContextAttributeEvent;
```

```
import javax.servlet.ServletContextAttributeListener;

public class SCAListener implements ServletContextAttributeListener {

    //添加域属性触发的方法
    public void attributeAdded(ServletContextAttributeEvent scab) {
        System.out.println("当前域属性已经加入到ServletContext中"+scab.getName());
    }

    //删除域属性触发的方法
    public void attributeRemoved(ServletContextAttributeEvent scab) {
        System.out.println("当前域属性已经被移除ServletContext中"+scab.getName());

    }

    //替换域属性触发的方法
    public void attributeReplaced(ServletContextAttributeEvent scab) {
        System.out.println("当前域属性已经被替换ServletContext中"+scab.getName()+scab.getValue());

    }

}
```

ii. 配置web.xml文件

```
<listener>
        <listener-class>cn.tedu.listener.SCAListener</listener-class>
 </listener>
```

c. 监听JavaBean在Session域中状态变化的监听器

```
HttpSessionBindingListener
HttpSessionActivationListener
```

i. HttpSessionBindingListener

让javabean自己感知到自己被存入或移除出session的状态变化的监听器

此监听器不需要单独开发类，也不需要在web.xml中进行配置，而是直接让需要监听的JavaBean实现即可。

其中具有两个方法：

```
//当当前javabean的对象被存入session时触发
public void valueBound(HttpSessionBindingEvent event) {


}
//当当前javabean的对象被移除出session时触发
public void valueUnbound(HttpSessionBindingEvent event) {
}
```

代码实现：

```
package cn.tedu.domain;

import javax.servlet.http.HttpSessionBindingEvent;
import javax.servlet.http.HttpSessionBindingListener;

public class User implements HttpSessionBindingListener{
    private String username;
    private int age;
    public User(String username, int age) {
```

```
        super();
        this.username = username;
        this.age = age;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

    //将当前javabea对象加入session域中时触发
    public void valueBound(HttpSessionBindingEvent event) {
        System.out.println("对象加入session域"+event.getName());
    }

    //将当前javabea对象移除session域中时触发
    public void valueUnbound(HttpSessionBindingEvent event) {
        System.out.println("对象移除session域"+event.getName());

    }


}
```

ii. *了解*HttpSessionActivationListener

让javabean自己感知到自己随着session被钝化和活化