Filter过滤器概述

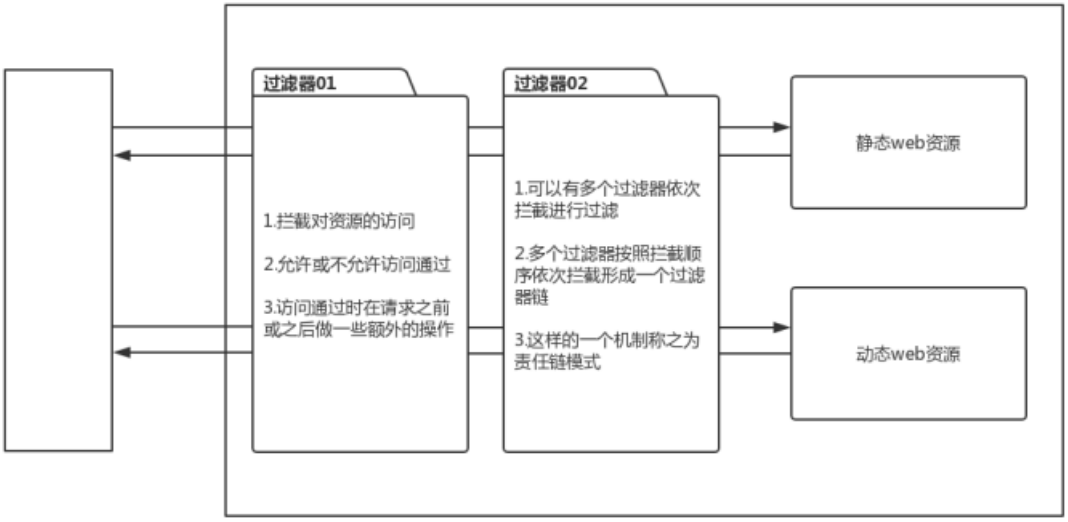
2019年3月1日 17:25

1. Filtert概述

Filter技术，即过滤器技术，是Servlet三大核心技术之一，主要用于拦截请求进行过滤操作，是实现例如权限控制、全站乱码解决、压缩响应等功能的基础，是web开发中最实用的技术之一。

过滤器可以拦截对资源的访问，拦截之后，可以控制访问是否放行，如果放行还可以在访问之前或之后做一些额外的操作。

过滤器可以配置多个，多个过滤器之前采用责任链模式，依次进行拦截。只有所有过滤器都通过，才可以访问到最终的资源。



2. 过滤器开发

a. 开发Filter的步骤
   i. 写一个类实现Filter接口
   ii. 在web.xml中配置Filter

b. 写一个类实现Filter接口

Filter接口

Method Summary	
void	destroy()

	Called by the web container to indicate to a filter that it is being taken out of service. 销毁的方法，当filter在web容器中被销毁时会调用此方法
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. 当有请求访问资源被当前过滤器拦截时，此方法执行。 request 代表当前拦截下来的请求对象 response代表当前拦截下来的响应对象 chain代表当前整个过滤器链 提供了doFilter()方法，可以放行过滤器
void	**init**(FilterConfig filterConfig) Called by the web container to indicate to a filter that it is being placed into service. 初始化的方法，当filter在web容器中被初始化时会调用此方法 filterConfig 代表当前Filter在web.xml中的配置信息

案例：自定义一个过滤器

```
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初始化了...");
    }

    public void doFilter(ServletRequest request,
    ServletResponse response,FilterChain chain) throws
    IOException, ServletException {
        System.out.println("进入FirstFilter...");
    }

    public void destroy() {
        System.out.println("FirstFilter销毁了...");
    }
```

```
}
```

c. 在web.xml中配置Filter

在web.xml中配置过滤器的过程非常类似于配置一个Servlet，只是要通过<filter><filter-mapping>标签实现配置。

其中<filter-mapping>中可以通过<servlet-name>或<url-pattern>来指定拦截的内容。<servlet-name>可以配置要拦截哪个名字的Servlet。<url-pattern>可以配置要拦截哪个url路径，配置方式和Servlet配置中的<url-pattern>一致。

一个web资源也可以被多个过滤器拦截，拦截顺序取决于过滤器的<filter-mapping>在web.xml中的配置顺序。

案例：配置过滤器

```
<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>
```

3. 过滤器的生命周期

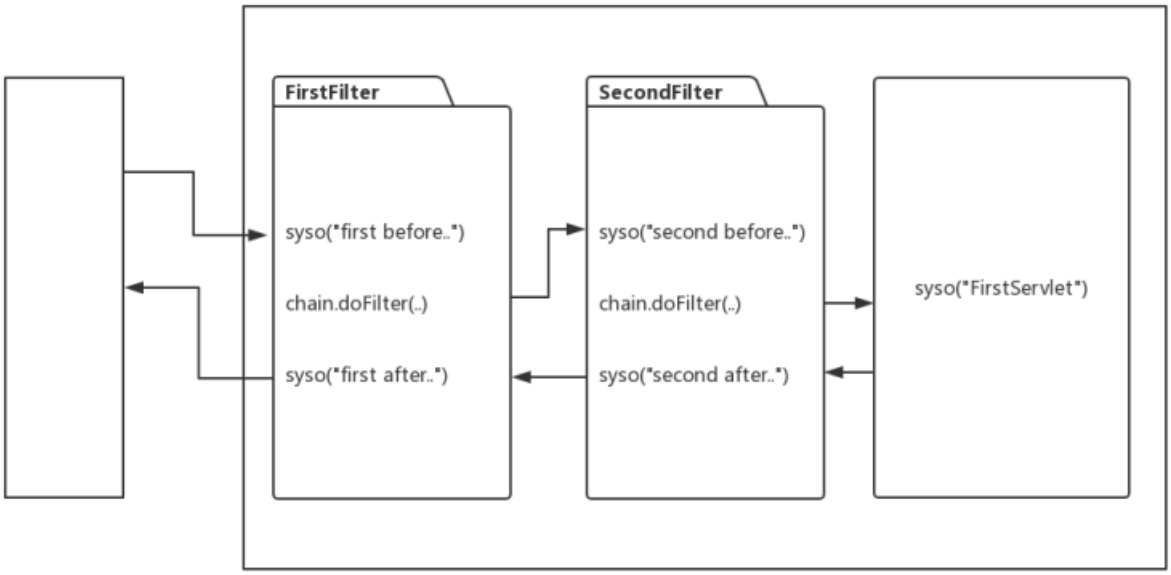
当web应用被加载到容器中时，过滤器对象被创建出来，立即执行init方法进行初始化，之后一直驻留在内存中为后续的拦截进行服务，拦截到任何资源都会执行doFilter方法执行拦截逻辑，可以在doFilter方法中可以通过FilterChain对象是否调用doFilter方法来控制是否允许通过，或者在允许通过时，在访问之前或之后做一些额外的操作。直到web应用移除出容器时，该web应用内部的Filter也跟着被销毁，在销毁之前会调用destory方法执行善后工作。

4. 多个过滤器执行过程

多个过滤器执行的先后顺序取决于过滤器的<filter-

mapping>的配置顺序，先配置的先拦截先执行过滤。

多个过滤器的执行过程，非常类似于方法调用的过程，每当调用chain.doFilter()都会执行后续资源，后续资源执行过后，在返回到当前过滤器执行chain.doFilter()之后的内容。



Filter案例 - 改造EasyMall

2019年4月6日 18:26

1. 全站乱码解决过滤器

web应用中，获取请求参数和发送响应数据时都有可能产生乱码，需要手动解决。

与其在每个Servlet中单独解决乱码，不如开发一个过滤器拦截所有资源，统一进行乱码解决。

a. 开发过滤器，进行乱码解决

i. 开发EncodingFilter

```
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;

public class EcodingFilter implements Filter {

    public void init(FilterConfig filterConfig) throws
    ServletException {
    }

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

    public void destroy() {
    }
```

```
}
```

ii. 配置该Filter，拦截所有的资源

```
<filter>
    <filter-name>EcodingFilter</filter-name>
    <filter-class>
    com.easymall.filter.EcodingFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>EcodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

iii. 解决响应乱码

其中采用的编码从web.xml中读取

```
<context-param>
    <param-name>encode</param-name>
    <param-value>utf-8</param-value>
</context-param>
```

```
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;

public class EcodingFilter implements Filter {
    private String encode = null;

    public void init(FilterConfig filterConfig) throws
    ServletException {
        encode =
        filterConfig.getServletContext().getInitParamet
        er("encode");
```

```
        }

        public void doFilter(ServletRequest request,
        ServletResponse response,FilterChain  chain)
        throws IOException, ServletException {
            //--全局响应乱码解决
            response.setContentType("text/html;charset="
            +encode);
            //--放行资源
            chain.doFilter(myRequest, response);
        }

        public void destroy() {

        }
}
```

b. 解决请求乱码

如果是POST请求，通过request.setCharacterEncoding就可以解决

但，如果是GET请求乱码，request.setCharacterEncoding无效，需要手动编解码解决乱码，但在过滤器中，可以获取所有请求参数解决乱码，但无法将请求参数重新设置回request对象。

所以换一种思路，通过装饰设计模式，改造request中和获取请求参数相关的

request.getparameterMap

request.getParameterValues

request.getParameter方法，其中增加乱码解决相关逻辑。

开发装饰者：

```
package com.easymall.filter;

import java.util.HashMap;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletRequestWrapper;

public class MyHttpServletRequest extends
HttpServletRequestWrapper {
    private String encode = null;
    private HttpServletRequest request = null;
    public MyHttpServletRequest(String
    encode,HttpServletRequest request) {
        super(request);
        this.encode = encode;
        this.request = request;
    }

    @Override
    public Map<String,String[]> getParameterMap() {
        try {
            //获取真正的Map，其中有乱码
            Map<String,String[]> map =
            request.getParameterMap();
            //建立目标Map，用于保存解决完乱码的请
            求参数
            Map<String,String[]> rmap = new
            HashMap<String, String[]>();
            //遍历真正的Map，解决乱码，存入目标
            map
            for(Map.Entry<String, String[]> entry :
            map.entrySet()){
                String key = entry.getKey();
                String vs [] = entry.getValue();
                String rvs [] = new String[vs.length];
                for(int i = 0;i<vs.length;i++){
                    rvs[i] = new
                    String(vs[i].getBytes("iso8859-1"
                    ),encode);
                }
                rmap.put(key, rvs);
            }
            //目标map中已经存放了所有解决完乱码的请
            求参数，将此map返回
            return rmap;
        } catch (Exception e) {
            e.printStackTrace();
```

```
                throw new RuntimeException(e);
            }
        }

        @Override
        public String[] getParameterValues(String name) {
            Map<String, String[]> rmap =
            getParameterMap();
            return rmap.get(name);
        }

        @Override
        public String getParameter(String name) {
            String[] values = getParameterValues(name);
            return values == null ? null : values[0];
        }
}
```

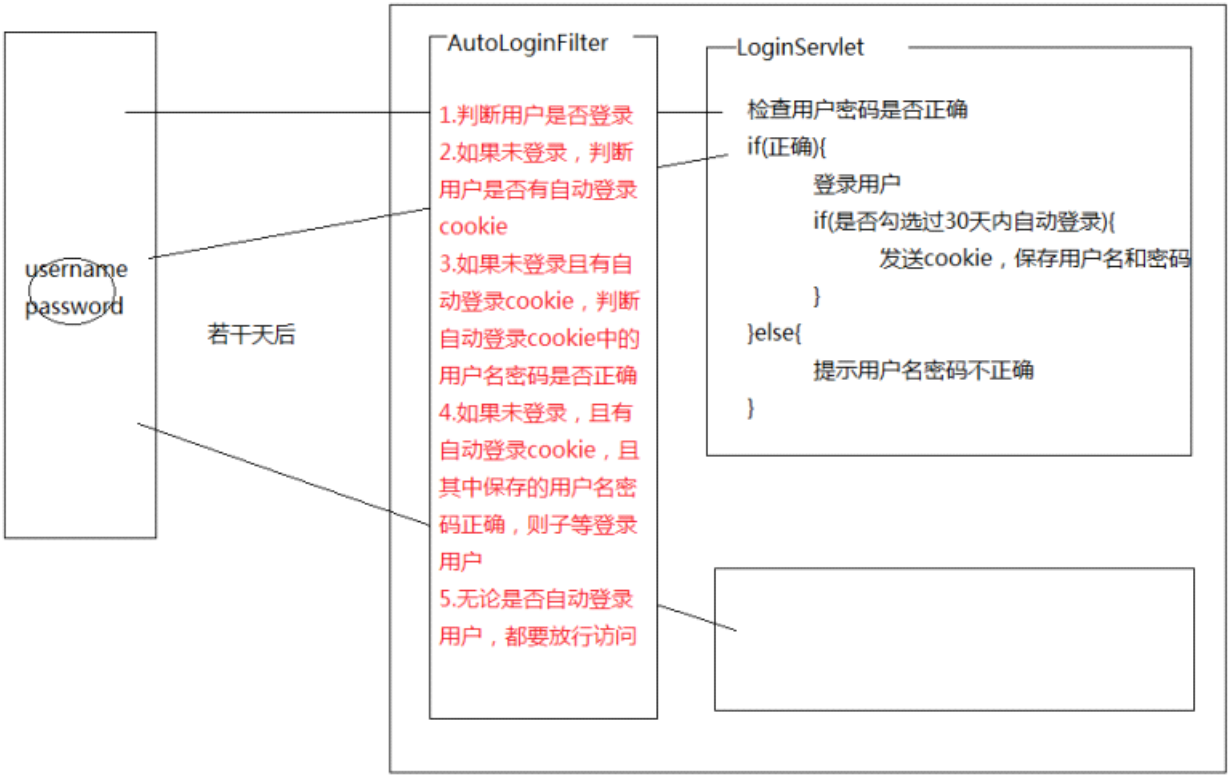
改造乱码解决过滤器，将request进行装饰，解决全站乱码，此处额外配置了一个开关，可以根据web.xml中的配置决定是否使用全站request乱码解决。如果当前tomcat默认的请求处理编码为iso8859-1,则开启此开关，自动解决全站乱码，如果当前tomcat默认的请求处理编码为utf-8，则关闭此开关，使用原有的request。

```
<context-param>
    <description>是否启用全局请求乱码解决
    </description>
    <param-name>use_request_encoder</param-
    name>
    <param-value>true</param-value>
</context-param>
```

```
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;

public class EcodingFilter implements Filter {
	private String encode = null;
	private boolean use_request_encoder = false;

	public void init(FilterConfig filterConfig) throws
	ServletException {
		encode =
		filterConfig.getServletContext().getInitParamet
		er("encode");
		use_request_encoder =
		Boolean.parseBoolean(filterConfig.getServletC
		ontext().getInitParameter("use_request_encod
		er"));
	}

	public void doFilter(ServletRequest request,
	ServletResponse response,FilterChain chain)
	throws IOException, ServletException {
		//--全局请求乱码解决
		ServletRequest myRequest =
		use_request_encoder ? new
		MyHttpServletRequest(encode,
		(HttpServletRequest) request) : request;
		//--全局响应乱码解决
		response.setContentType("text/html;charset="
		+encode);
		//--放行资源
		chain.doFilter(myRequest, response);
	}

	public void destroy() {

	}

}
```

2. 30天内自动登录过滤器



a. 改造LoginServlet

```
package com.easymall.web;

import java.io.IOException;
import java.net.URLEncoder;

import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.easymall.domain.User;
import com.easymall.exception.MsgException;
import com.easymall.service.UserService;

public class LoginServlet extends HttpServlet {



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

```
//1.获取参数
String username =
request.getParameter("username");
String password =
request.getParameter("password");
String remname =
request.getParameter("remname");
//--处理记住用户名功能
if("true".equals(remname)){//如果remname为"true",
则证明记录用户名被选中
    //利用cookie存储用户名
    Cookie cookie = new
    Cookie("username",URLEncoder.encode(usern
    ame, "utf-8"));
    //设置最大生存时间，将cookie保存在本地磁盘
    cookie.setMaxAge(3600*24*30);//秒值
    //设置path
    cookie.setPath(request.getContextPath()+"/");
    response.addCookie(cookie);
}else{
    Cookie cookie = new Cookie("username","");
    //设置最大生存时间，将cookie保存在本地磁盘
    cookie.setMaxAge(0);//秒值
    //设置path
    cookie.setPath(request.getContextPath()+"/");
    response.addCookie(cookie);

}

//2.调用Service来实现登录用户
User user = null;
try{
    UserService service = new UserService();
    user = service.loginUser(username,password);
}catch (MsgException e) {
    //--用户名密码不正确，回到页面提示
    request.setAttribute("msg", e.getMessage());
    request.getRequestDispatcher("/login.jsp").for
    ward(request, response);
    return;
}
```

```
            //--用户名密码正确，登录用户
            request.getSession().setAttribute("user", user);

            //--处理30天内自动登录
            String autologin =
            request.getParameter("autologin");
            if("true".equals(autologin)){
                //--勾选过30天内自动登录，则发送cookie保存用
                户名 密码
                Cookie autologinc = new
                Cookie("autologin",URLEncoder.encode(usern
                ame,"utf-8")+"#"+password);
                autologinc.setPath("/");
                autologinc.setMaxAge(60 * 60 * 24 * 30);
                response.addCookie(autologinc);
            }

            //跳转回主页
            response.sendRedirect("/index.jsp");
        }


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

        }

}
```

b. 开发AutoLoginFilter

```
package com.easymall.filter;

import java.io.IOException;
import java.net.URLDecoder;

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.Cookie;
import javax.servlet.http.HttpServletRequest;

import com.easymall.domain.User;
import com.easymall.exception.MsgException;
import com.easymall.service.UserService;

public class AutoLoginFilter implements Filter {

    private UserService userService = new UserService();

    public void init(FilterConfig filterConfig) throws
    ServletException {

    }

    public void doFilter(ServletRequest request,
    ServletResponse response,FilterChain chain) throws
    IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest)
        request;
        //1.用户未登录
        if(req.getSession(false) == null
            || req.getSession().getAttribute("user") ==
            null){
            //2.带了自动登录cookie
            Cookie[] cs = req.getCookies();
            Cookie findc = null;
            if(cs!=null){
                for(Cookie c : cs){
                    if("autologin".equals(c.getName())){
                        findc = c;
                    }
                }
            }
            if(findc != null){
                //3.自动登录cookie中的用户名密码正确
                String kv [] = findc.getValue().split("#");
                String username =
                URLDecoder.decode(kv[0], "utf-8");
                String password = kv[1];
```

```
                    try {
                        User user =
                        userService.loginUser(username,
                        password);
                        //4.都成立就自动登录
                        req.getSession().setAttribute("user",
                        user);
                    } catch (MsgException e) {
                        //4.用户名密码不正确，则不进行自动登
                        录，但是也不抛异常
                    }
                }
            }
            //5.无论是否自动登录都放行资源
            chain.doFilter(request, response);
        }

        public void destroy() {

        }

}
```

c. 改造LogoutServlet

需要在登出操作时，删除30天内自动登录cookie

```
package com.easymall.web;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LogoutServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
        //1.杀死session，用户登出
        if(request.getSession(false) != null){
```

```
                request.getSession().invalidate();
            }
            //2.删除30天内自动登录cookie
            Cookie autologinc = new Cookie("autologin","");
            autologinc.setPath("/");
            autologinc.setMaxAge(0);
            response.addCookie(autologinc);
            //3.重定向回主页
            response.sendRedirect("/index.jsp");
        }

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

        }

}
```

3. 了解：过滤器的其他案例
    a. 粗粒度权限控制
    b. 全站响应压缩
    c. 敏感词过滤

MD5加密算法

2019年4月6日 18:26

1. 数据摘要算法

在企业级开发过程中，有很多重要信息，是需要加密后存储的。

目前在行业中，应用最广泛的加密算法是数据摘要算法 也称作数据指纹算法。

数据摘要算法有很多，目前比较流行的是 MD5 SHA1 SHA512等等。

课上使用MD5，其他算法其实也差不多。

数据摘要算法的特点：

任何二进制，经过MD5算法机密之后，可以得到一个128位的二进制值，通常写成32位十六进制。

原文相同，算出的MD5密文一定相同，原文不同，算出的MD5密文一定不同(可能重复，但概率非常低)。且，MD5值没有任何规律。

只能由明文算出密文，密文永远无法算回明文(实际上是可以算的，只是计算量太大，目前的计算机无法在可接受的的时间内完成)。

数据摘要算法的主要应用场景：

数据库中密码加密

下载文件时文件的校验

网盘的秒传

…

2. MD5的实现

a. mysql数据库中

MD5()函数可以实现md5算法

```
mysql> select md5(123);
+----------------------------------+
| md5(123)                         |
+----------------------------------+
| 202cb962ac59075b964b07152d234b70 |
+----------------------------------+
```

b. java中

```
package com.easymall.utils;

import java.math.BigInteger;
import java.security.MessageDigest;
import
java.security.NoSuchAlgorithmException;

public class MD5Utils {
    /**
     * 使用md5的算法进行加密
     */
    public static String md5(String plainText)
    {
        byte[] secretBytes = null;
        try {
            secretBytes =
            MessageDigest.getInstance("md5"
            )
                .digest(plainText.getBytes());
```

```
        } catch (NoSuchAlgorithmException e)
        {
            throw new RuntimeException("没
            有md5这个算法！");
        }
        String md5code = new BigInteger(1,
        secretBytes).toString(16);
        for (int i = 0; i < 32 -
        md5code.length(); i++) {
            md5code = "0" + md5code;
        }
        return md5code;
    }
}
```

3. 改造EasyMall使用MD5加密

a. 数据库中user表密码都要变为MD5加密

```
update user set password = MD5(password);
```

b. 改造EasyMall注册功能，密码加密后存储

改造RegistServlet，将密码加密后存储

```
package com.easymall.web;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import
javax.servlet.http.HttpServletResponse;

import com.easymall.domain.User;
import
com.easymall.exception.MsgException;
import com.easymall.service.UserService;
import com.easymall.utils.MD5Utils;
import com.easymall.utils.WebUtils;

public class RegistServlet extends HttpServlet
{

    public void doGet(HttpServletRequest
    request, HttpServletResponse response)
        throws ServletException, IOException
        {
        //1.获取请求参数
        String username =
        request.getParameter("username");
        String password =
        request.getParameter("password");
        String password2 =
        request.getParameter("password2");
        String nickname =
        request.getParameter("nickname");
        String email =
        request.getParameter("email");
```

```
String valistr =
request.getParameter("valistr");

//验证码 session
//从session中取出ValistrSevlet生成的验证码文本
String code = (String)
request.getSession().getAttribute("code");
//获取用户输入的验证码 --- 上方已经获取过了↑
//判断用户输入的数据和valistrServlet中的文本是否相同
if(!code.equalsIgnoreCase(valistr)){
    //验证码正确不需要任何操作，如果错误应该在页面中提示错误信息
    request.setAttribute("msg", "验证码不正确");
    request.getRequestDispatcher("/regist.jsp").forward(request,
    response);
    return;
}
//2.非空校验
if(WebUtils.isNull(username)){//利用工具类来校验
    //request可以用作域对象，
    //利用域对象身上的域将数据由当前
```

```
            servlet传递到regist.jsp中
            request.setAttribute("msg", "用户
            名不能为空");
            request.getRequestDispatcher("/re
            gist.jsp").forward(request,
            response);
            return;
        }
        if(WebUtils.isNull(password)){//利用工
        具类来校验
            //request可以用作域对象，
            //利用域对象身上的域将数据由当前
            servlet传递到regist.jsp中
            request.setAttribute("msg", "密码
            不能为空");
            request.getRequestDispatcher("/re
            gist.jsp").forward(request,
            response);
            return;
        }
        if(WebUtils.isNull(password2)){//利用
        工具类来校验
            //request可以用作域对象，
            //利用域对象身上的域将数据由当前
            servlet传递到regist.jsp中
            request.setAttribute("msg", "确认
            密码不能为空");
            request.getRequestDispatcher("/re
```

```
            gist.jsp").forward(request,
            response);
            return;
        }
        if(WebUtils.isNull(nickname)){//利用工
        具类来校验
            //request可以用作域对象，
            //利用域对象身上的域将数据由当前
            servlet传递到regist.jsp中
            request.setAttribute("msg", "昵称
            不能为空");
            request.getRequestDispatcher("/re
            gist.jsp").forward(request,
            response);
            return;
        }
        if(WebUtils.isNull(email)){//利用工具类
        来校验
            //request可以用作域对象，
            //利用域对象身上的域将数据由当前
            servlet传递到regist.jsp中
            request.setAttribute("msg", "邮箱
            不能为空");
            request.getRequestDispatcher("/re
            gist.jsp").forward(request,
            response);
            return;
        }
```

```
if(WebUtils.isNull(valistr)){//利用工具类
来校验
    //request可以用作域对象，
    //利用域对象身上的域将数据由当前
    servlet传递到regist.jsp中
    request.setAttribute("msg", "验证
    码不能为空");
    request.getRequestDispatcher("/re
    gist.jsp").forward(request,
    response);
    return;
}
//3.密码一致性校验
if(password.trim() !="" &&
password2.trim() != ""
    && !
    password.trim().equals(password2.
    trim())){
    request.setAttribute("msg", "两次
    密码不一致");
    request.getRequestDispatcher("/re
    gist.jsp").forward(request,
    response);
    return;
}
//4.邮箱格式校验
String reg = "\\w+@\\w+(\\.\\w+)+";
if(!email.matches(reg)){
```

```
            request.setAttribute("msg", "邮箱
            格式不正确");
            request.getRequestDispatcher("/re
            gist.jsp").forward(request,
            response);
            return;
        }

        //5.调用Service完成用户注册功能
        //--封装数据到bean
        User user = new
        User(0,username,MD5Utils.md5(passw
        ord),nickname,email);
        //--调用Service中的相关方法
        UserService service = new
        UserService();
        try{
            service.registUser(user);
        }catch (MsgException e) {
            //--底层出问题要提示用户 将错误信
            息展示回原来的页面
            request.setAttribute("msg",
            e.getMessage());
            request.getRequestDispatcher("/re
            gist.jsp").forward(request,
            response);
            return;
        }
```

```
            //6.利用定时刷新返回首页
            response.getWriter().write("<h1
            align='center'><font color='red'>恭喜
            注册成功，3秒后跳转回首页！</font>
            </h1>");
            response.setHeader("refresh",
            "3;url=http://www.easymall.com");
        }

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

        }

}
```

c. 改造EasyMall登录功能，密码要机密后再校验

改造LoginServlet，将密码加密后校验

```
package com.easymall.web;

import java.io.IOException;
import java.net.URLEncoder;

import javax.servlet.ServletException;
```

```
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import
javax.servlet.http.HttpServletResponse;

import com.easymall.domain.User;
import
com.easymall.exception.MsgException;
import com.easymall.service.UserService;
import com.easymall.utils.MD5Utils;

public class LoginServlet extends HttpServlet
{



    public void doGet(HttpServletRequest
    request, HttpServletResponse response)
        throws ServletException, IOException
        {
        //1.获取参数
        String username =
        request.getParameter("username");
        String password =
        request.getParameter("password");
        String remname =
        request.getParameter("remname");
```

```
//--处理记住用户名功能
if("true".equals(remname)){//如果
remname为"true",则证明记录用户名被
选中
    //利用cookie存储用户名
    Cookie cookie = new
    Cookie("username",URLEncoder.e
    ncode(username, "utf-8"));
    //设置最大生存时间，将cookie保存
    在本地磁盘
    cookie.setMaxAge(3600*24*30);//
    秒值
    //设置path
    cookie.setPath(request.getContext
    Path()+"/");
    response.addCookie(cookie);
}else{
    Cookie cookie = new
    Cookie("username","");
    //设置最大生存时间，将cookie保存
    在本地磁盘
    cookie.setMaxAge(0);//秒值
    //设置path
    cookie.setPath(request.getContext
    Path()+"/");
    response.addCookie(cookie);

}
```

```
//2.调用Service来实现登录用户
User user = null;
try{
    UserService service = new
    UserService();
    user =
    service.loginUser(username,MD5U
    tils.md5(password));
}catch (MsgException e) {
    //--用户名密码不正确，回到页面提
    示
    request.setAttribute("msg",
    e.getMessage());
    request.getRequestDispatcher("/lo
    gin.jsp").forward(request,
    response);
    return;
}

//--用户名密码正确，登录用户
request.getSession().setAttribute("use
r", user);

//--处理30天内自动登录
String autologin =
request.getParameter("autologin");
if("true".equals(autologin)){
```

```
            //--勾选过30天内自动登录，则发送
            cookie保存用户名 密码
            Cookie autologinc = new
            Cookie("autologin",URLEncoder.en
            code(username,"utf-8")+"#"+pass
            word);
            autologinc.setPath("/");
            autologinc.setMaxAge(60 * 60 * 24
            * 30);
            response.addCookie(autologinc);
        }

        //跳转回主页
        response.sendRedirect("/index.jsp");
    }


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

    }

}
```

d. 改造30天内自动登录，使用MD5

改造LoginServlet，自动登录时向cookie中存储的密码需要是MD5格式

```
package com.easymall.web;

import java.io.IOException;
import java.net.URLEncoder;

import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import
javax.servlet.http.HttpServletResponse;

import com.easymall.domain.User;
import
com.easymall.exception.MsgException;
import com.easymall.service.UserService;
import com.easymall.utils.MD5Utils;

public class LoginServlet extends HttpServlet
{



    public void doGet(HttpServletRequest
    request, HttpServletResponse response)
```

```
throws ServletException, IOException
{
//1.获取参数
String username =
request.getParameter("username");
String password =
request.getParameter("password");
String remname =
request.getParameter("remname");
//--处理记住用户名功能
if("true".equals(remname)){//如果
remname为"true",则证明记录用户名被
选中
    //利用cookie存储用户名
    Cookie cookie = new
    Cookie("username",URLEncoder.e
    ncode(username, "utf-8"));
    //设置最大生存时间，将cookie保存
    在本地磁盘
    cookie.setMaxAge(3600*24*30);//
    秒值
    //设置path
    cookie.setPath(request.getContext
    Path()+"/");
    response.addCookie(cookie);
}else{
    Cookie cookie = new
    Cookie("username","");
```

```
            //设置最大生存时间，将cookie保存
            在本地磁盘
            cookie.setMaxAge(0);//秒值
            //设置path
            cookie.setPath(request.getContext
            Path()+"/");
            response.addCookie(cookie);

        }

        //2.调用Service来实现登录用户
        User user = null;
        try{
            UserService service = new
            UserService();
            user =
            service.loginUser(username,MD5U
            tils.md5(password));
        }catch (MsgException e) {
            //--用户名密码不正确，回到页面提
            示
            request.setAttribute("msg",
            e.getMessage());
            request.getRequestDispatcher("/lo
            gin.jsp").forward(request,
            response);
            return;
        }
```

```
        //--用户名密码正确，登录用户
        request.getSession().setAttribute("use
        r", user);

        //--处理30天内自动登录
        String autologin =
        request.getParameter("autologin");
        if("true".equals(autologin)){
            //--勾选过30天内自动登录，则发送
            cookie保存用户名 密码
            Cookie autologinc = new
            Cookie("autologin",URLEncoder.en
            code(username,"utf-8")+"#"+MD5
            Utils.md5(password));
            autologinc.setPath("/");
            autologinc.setMaxAge(60 * 60 * 24
            * 30);
            response.addCookie(autologinc);
        }

        //跳转回主页
        response.sendRedirect("/index.jsp");
    }


    public void doPost(HttpServletRequest
    request, HttpServletResponse response)
```

```
        throws ServletException, IOException
        {
        doGet(request, response);

    }

}
```

在过滤器中校验密码也要是MD5格式

**直接从cookie中获取的密码就是MD5格式的，不需要改造过滤器。

Listener监听器

2019年4月6日 18:27

1. Listener监听器概述

Listener监听器是JAVAEE中三大核心技术之一。

Listener监听器可以用来监听web应用在工作时发生的各种事件，来进行响应的处理。

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
```

案例：

```
package cn.tedu.listener;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class MyServletContextListener  implements
ServletContextListener  {

    public void contextInitialized(ServletContextEvent  sce) {
        System.out.println("ServletContext被初始化
        了。。。"+sce.getServletContext());
    }

    public void contextDestroyed(ServletContextEvent  sce) {
        System.out.println("ServletContext要被销毁
        了。。。"+sce.getServletContext());
    }
```

```
}
```

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

b. 监听三大作用域中属性变化的监听器

```
ServletContextAttributeListener
HttpSessionAttributeListener
ServletRequestAttributeListener
```

```
package cn.tedu.listener;

import javax.servlet.ServletContextAttributeEvent;
import javax.servlet.ServletContextAttributeListener;

public class MySCAtrrListener implements
ServletContextAttributeListener  {

    public void attributeAdded(ServletContextAttributeEvent
    scab) {
        System.out.println("属性被加入sc
        域"+scab.getName()+"~"+scab.getValue());
    }

    public void
    attributeRemoved(ServletContextAttributeEvent  scab) {
        System.out.println("属性被移除出sc
        域"+scab.getName()+"~"+scab.getValue());
    }

    public void
    attributeReplaced(ServletContextAttributeEvent  scab) {
        System.out.println("属性被替换在sc
        域"+scab.getName()+"~"+scab.getValue()+"~"+sca
        b.getServletContext().getAttribute(scab.getName()));
```

```
    }

}
```

```
<listener>
    <listener-class>
    cn.tedu.listener.MySCAtrrListener</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) {
}
```

ii. *了解*HttpSessionActivationListener

让javabean自己感知到自己随着session被钝化和活化

3. 监听器案例

开发监听器，记录应用启动关闭 记录资源被访问 记录用户登录注销 的日志

Log4j日志框架

2019年4月6日 18:29

1. Log4j概述

Log4j即log for java，是专门用来在java中记日志的工具，简单易用，使用便捷，在java开发中应用非常广泛，基本上是目前java记日志的事实上的通用工具。

Log4j中的日志级别

可以在使用log4j输出日志时，选择日志的不同级别，建议根据推荐选择合适的级别输出日志，方便统一管理。

可以在log4j的配置文件中配置输出哪个级别的日志，则高于或等于这个级别的日志会被输出，低于这个级别的日志会被忽略。

off 最高等级， 用于关闭所有日志记录。
fatal 指出每个严重的错误事件将会导致应用程序的退出。
error 指出虽然发生错误事件， 但仍然不影响系统的继续运行。
warn 表明会出现潜在的错误情形。
info 一般和在粗粒度级别上， 强调应用程序的运行全程。
debug 一般用于细粒度级别上， 对调试应用程序非常有帮助。
all 最低等级， 用于打开所有日志记录

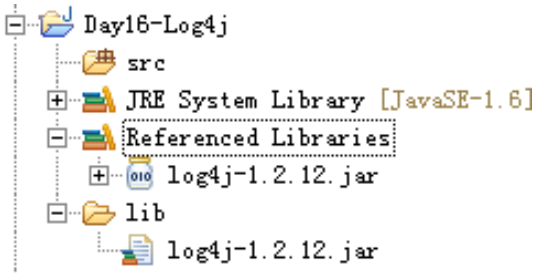
2. Log4j的使用

a. 下载log4j的jar包

log4j并不是j2se的一部分，需要使用时，需要导入第三方开发包。

http://jakarta.apache.org/log4j

b. 在项目中导入log4j相关的jar



c. 在项目的类加载目录下放置一个log4j.properties配置文件

在该文件中进行log4j相关的配置，通常不会真的自己配置，而是从模板配置中拷贝。

```
###设置###
log4j.rootLogger=debug,stdout,D,E
###输出信息到控制抬###
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%-5p]%d{yyyy-MM-ddHH:mm:ss,SSS}method:%l%n%m%n
###输出DEBUG级别以上的日志到=E://logs/error.log###
log4j.appender.D=org.apache.log4j.DailyRollingFileAppender
log4j.appender.D.File=E://logs/log.log
log4j.appender.D.Append=true
log4j.appender.D.Threshold=DEBUG
log4j.appender.D.layout=org.apache.log4j.PatternLayout
log4j.appender.D.layout.ConversionPattern=%-d{yyyy-MM-ddHH:mm:ss}[%t:%r]-[%p]%m%n
###输出ERROR级别以上的日志到=E://logs/error.log###
log4j.appender.E=org.apache.log4j.DailyRollingFileAppender
log4j.appender.E.File=E://logs/error.log
log4j.appender.E.Append=true
log4j.appender.E.Threshold=ERROR
log4j.appender.E.layout=org.apache.log4j.PatternLayout
log4j.appender.E.layout.ConversionPattern=%-d{yyyy-MM-ddHH:mm:ss}[%t:%r]-[%p]%m%n
```

i. appender配置

org. apache. log4j. ConsoleAppender（控制台），
org. apache. log4j. FileAppender（文件），
org. apache. log4j. DailyRollingFileAppender（每天产生一个日 志文件），
org. apache. log4j. RollingFileAppender（文件大小到达指定尺寸的时候产生一个新的文件），
org. apache. log4j. WriterAppender（将日 志信息以流格式发送到任意指定的地方）

ii. layout配置

org. apache. log4j. HTMLLayout（以HTML表格形式布局），
org. apache. log4j. PatternLayout（可以灵活地指定布局模式），
org. apache. log4j. SimpleLayout（包含日 志信息的级别和信息字符串），
org. apache. log4j. TTCCLayout（包含日 志产生的时间、 线程、 类别等等信息）

iii. Pattern配置

如果选择PatternLayout就需要自己配置日志格式，这是通过如下占位符实现的

%p 输出优先级， 即DEBUG， INFO， WARN， ERROR， FATAL
%r 输出自 应用启动到输出该log信息耗费的毫秒数
%c 输出所属的类目， 通常就是所在类的全名
%t 输出产生该日 志事件的线程名
%n 输出一个回车换行符， Windows平台为“rn”， Unix平台为“n”
%d 输出日 志时间点的日 期或时间， 默认格式为ISO8601， 也可以在其后指定格式， 比如： %d{yyy MMM dd HH: mm: ss, SSS}， 输出类似
： 2002年10月 18日 22： 10： 28， 921
%l 输出日志事件的发生位置， 包括类目 名、 发生的线程， 以及在代码中的行数。 举例： Testlog4. main(TestLog4. java: 10)

d. log4j的javaapi

i. 获取日志记录器

通过Logger类提供静态方法可以通过名字获取一个日志记录器，每个日志记录器都要有一个名字，如果该名字的日志记录器不存在则创建。

```
public static Logger getLogger(String name)
public static Logger getLogger(Class clz)
```

ii. 配置log4j

可以通过api配置，如果不配置，默认会在类加载目录下寻找log4j.properties作为配置。

iii. 记录日志

日志记录器提供了不同方法用于记录不同级别的日志，在开发中，根据要记录的日志的紧要程度选择不同的方法记录日志。

```
Logger. debug ( Object message ) ;
Logger. info ( Object message ) ;
Logger. warn ( Object message ) ;
Logger. error ( Object message ) ;
Logger. fatal( Object message ) ;
```