# 过滤器

过滤器,设计执行流程:

1. 用户访问服务器
2. 过滤器:对Servlet进行拦截
3. 先进入过滤器,过滤器处理
4. 过滤器处理完后,在放行,此时请求到达Servlet/Jsp
5. Servlet处理
6. Servlet处理完,再回到过滤器,最后由tomcat服务器响应用户

import javax.servlet.\*;

接口Fileter (过滤器也是一个Servlet)

开发步骤:

1. 写一个普通java类,实现Filter接口
2. 配置过滤器

范例:

|  |
| --- |
| package filter;  import javax.servlet.\*;  import java.io.IOException;  public class HelloFilter implements Filter {  public HelloFilter(){  System.out.println("0.创建过滤器实例");  }  @Override  public void init(FilterConfig filterConfig) throws ServletException {  System.out.println("1.执行过滤器初始化方法");  }  //过滤器业务处理方法,在请求到达servlet之前,先进入此方法处理公用的业务逻辑操作  @Override  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  System.out.println("2.执行过滤器处理方法");  //如果有下一个过滤器,进入下一个过滤器,否则执行访问的servlet  filterChain.doFilter(servletRequest,servletResponse);  System.out.println("4.Servlet处理完成,又回到过滤器");  }  @Override  public void destroy() {  System.out.println("5.销毁过滤器实例");  }  } |

web.xml中配置:

|  |
| --- |
| <!--过滤器配置-->  <filter>  <!--过滤器名称-->  <filter-name>HelloFilter</filter-name>  <!--过滤器类-->  <filter-class>filter.HelloFilter</filter-class>  </filter>  <filter-mapping>  <filter-name>HelloFilter</filter-name>  <!--拦截的url-->  <url-pattern>/\*</url-pattern>  </filter-mapping> |

# 2.相关的api

interface Filter 过滤器核心接口

void init(FilterConfig);初始化方法,在服务器启动时执行

void doFilter(request,response,filterChain);过滤器拦截的业务处理方法

|  |
| --- |
| public void doFilter(xxx){   1. 过滤请求 2. chain.doFilter(request,response) 3. 过滤响应   } |

void destory();销毁过滤器实例时候调用

interface FilterConfig 获取初始化参数信息

Stirng getInitParameter(String name) //获取参数名对应的值

Enumeration getInitParameterNames() //获取一个参数名集合的迭代器

interface FilterChain 过滤器链参数,一个个过滤器形成一个执行链.

void doFilter(ServletRequest request,ServletResponse response)//执行下一个过滤器或者Servlet/Jsp

# 拦截

注意:拦截是有顺序的,根据web.xml中的从上到下的顺序

<filter-mapping>

...

url-pattern可以写多个

1.拦截所有

<url-pattern>/\*</url-pattern>

2.拦截指定jsp

<url-pattern>/index.jsp</url-pattern>

<url-pattern>/list.jsp</url-pattern>

3.拦截所有的jsp

<url-pattern>\*.jsp</url-pattern>

4根据servlet的内部名称拦截

<url-pattern>HelloServlet</url-pattern>

5.拦截指定的servlet

<url-pattern>/hello</url-pattern>

6.指定拦截指定的类型,可以写多个

//默认拦截的类型

<dispatcher>REQUEST</dispatcher>

//拦截转发

<dispatcher>FORWARD</dispatcher

//拦截包含的页面(RequestDispatcher.include(/page.jsp);)对page.jsp也进行拦截

<dispatcher>INCLUDE</dispatcher>

//拦截声明式异常信息:

<dispatcher>ERROR</dispatcher>

</filter-mapping>

# 4.案例

## 4.1编码统一处理

几乎每一个Servlet都要涉及编码处理,把公用代码抽取出来,用过滤器处理

get方式对中文没有乱码

post方式是有乱码的

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <title>Title</title>  </head>  <body>  <form method="post" action="/web/getparam">  用户名:<input type="text" name="userName">  <input type="submit" value="提交">  </form>  </body>  </html> |

|  |
| --- |
| public class GetParamServlet 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 {  String name = request.getParameter("userName");  System.out.println(name);  }  } |

|  |
| --- |
| /\*\*  \* 参数中文乱码问题的过滤器  \*/  public class EncodingFilter implements Filter {  @Override  public void init(FilterConfig filterConfig) throws ServletException {  }  @Override  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  HttpServletRequest request = (HttpServletRequest)servletRequest;  if("POST".equals(request.getMethod())){  request.setCharacterEncoding("utf-8");  }  filterChain.doFilter(request,servletResponse);  }  @Override  public void destroy() {  }  } |

## 4.2压缩网页内容

用户浏览一个网页,服务器->发送一个网页给用户

使用**java.util.zip.GZIPOutputStream类对网页进行压缩**

定义:

java.lang.Object

java.io.OutputStream

java.io.FilterOutputStream

java.util.zip.DeflaterOutputStream

java.util.zip.GZIPOutputStream

由于:response.getWriter()返回的PrintWriter是不带缓冲区的,通过write是直接输出到网页上的,而不是到缓冲区中,所以使用过滤器无法过滤,我们需要改造response对象,让它能够返回一个带缓冲区的PrintWriter

PrinitWriter

PrintWriter的构造函数:

1. PrintWriter(File file) 没缓冲区
2. PrintWriter(String fileName) 没缓冲区
3. PrintWriter(Writer out) 有缓冲区,即out为缓冲区
4. PrintWriter(OutputStream out) 有缓冲区,即out为缓冲区

ContentServlet.java

|  |
| --- |
| package 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 = "ContentServlet")  public class ContentServlet extends HttpServlet {  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  this.doGet(request, response);  }  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  //准备内容  StringBuffer sb = new StringBuffer();  for(int i=1;i<=3000;i++){  sb.append("abcd");  }  response.getOutputStream().write(sb.toString().getBytes());  }  } |

ContentGZIPFilter.java

|  |
| --- |
| package filter;  import javax.servlet.\*;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.http.HttpServletResponseWrapper;  import java.io.ByteArrayOutputStream;  import java.io.CharArrayWriter;  import java.io.IOException;  import java.io.PrintWriter;  import java.util.zip.GZIPOutputStream;  public class ContentGZIPFilter implements Filter {  @Override  public void init(FilterConfig filterConfig) throws ServletException {  }  @Override  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  //1)过滤请求  //创建一个装饰者类  MyHttpResponse response = new MyHttpResponse((HttpServletResponse) servletResponse);  //2)放行  filterChain.doFilter(servletRequest,response);  char[] content = response.getCharArray();  //gzip压缩  ByteArrayOutputStream buf = new ByteArrayOutputStream();  GZIPOutputStream gzip = new GZIPOutputStream(buf);  gzip.write(new String(content).getBytes());  gzip.finish();  byte[] result = buf.toByteArray();  //设置头部  response.setHeader("content-encoding","gzip");  //输出到浏览器  //response.getWriter();这个方法被重写了肯定无法输出到浏览器  response.getOutputStream().write(result);  }  @Override  public void destroy() {  }  }  class MyHttpResponse extends HttpServletResponseWrapper{  private HttpServletResponse response;  private CharArrayWriter charArrayWriter = new CharArrayWriter();  public MyHttpResponse(HttpServletResponse response) {  super(response);  this.response = response;  }  /\*\*  \* 重写getWriter()方法,让其返回带有缓冲区的PrintWriter  \*/  @Override  public PrintWriter getWriter() throws IOException {  /\*\*  \* 此时返回的是一个带有缓冲区的PrintWriter调用了write()方法,  \* 则数据会被写charArrayWriter  \*/  return new PrintWriter(this.charArrayWriter);  }  public char[] getCharArray(){  return this.charArrayWriter.toCharArray();  }  } |

## 4.3登陆权限

访问某个网页之前,通过过滤器进行权限的判断

比如你要访问某个需要登陆的index.jsp时,如果不用过滤器,那么就需要在index.jsp中判断是否登陆了,例如下面的代码:

<c:if test="${empty sessionScope.user}">

<c:redirect url="/login.jsp"></c:redirect>

</c:if>

如果有很多类似index.jsp的页面都需要这种登录的权限,那么每个页面都这一行这样的代码,这样太繁琐了,所以要使用过滤器,在登陆之前进行拦截,判断是否登陆了.

login.jsp:

|  |
| --- |
| <%@ page contentType="text/html;charset=UTF-8" language="java" %>  <html>  <head>  <title>登陆界面</title>  </head>  <body>  <font color="red">${msg}</font>  <form action="${pageContext.request.contextPath}/login" method="post">  用户名:<input type="text" name="userName"><br>  密码:<input type="password" name="userPwd"><br>  <input type="submit" value="登陆">  </form>  </body>  </html> |

LoginServlet.java:

|  |
| --- |
| 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 javax.servlet.http.HttpSession;  import java.io.IOException;  @WebServlet(name = "LoginServlet")  public class LoginServlet extends HttpServlet {  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  this.doGet(request, response);  }  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  //接受参数  String userName = request.getParameter("userName");  String userPwd = request.getParameter("userPwd");  //判断登陆  if("eric".equals(userName)&&"123456".equals(userPwd)){  //登陆成功  //把用户数据存到session域名对象中  //true表示有session则返回,没有则创建并返回一个session  HttpSession session = request.getSession(true);  session.setAttribute("user",userName);  //跳转到用户的主页  response.sendRedirect(request.getContextPath() + "/index.jsp");  }else {  //登陆失败  request.setAttribute("msg","用户名或者密码错误");  request.getRequestDispatcher("/login.jsp").forward(request,response);  }  }  } |

SecurityFilter.java:

|  |
| --- |
| package filter;  import javax.servlet.\*;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.http.HttpSession;  import java.io.IOException;  public class SecurityFilter implements Filter{  @Override  public void init(FilterConfig filterConfig) throws ServletException {  }  @Override  public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  //是否登陆判断逻辑  //先判断有无session对象存在  HttpServletRequest request = (HttpServletRequest)servletRequest;  HttpServletResponse response = (HttpServletResponse) servletResponse;  HttpSession session = request.getSession(false);  if(session == null){  //没有登陆成功  response.sendRedirect(request.getContextPath()+"/noAuth.html");  return;  }else{  String user = (String)session.getAttribute("user");  if(user==null){  //没有登陆成功  response.sendRedirect(request.getContextPath()+"/noAuth.html");  return;  }  }  //如果登陆成功,则方行  filterChain.doFilter(servletRequest, servletResponse);  }  @Override  public void destroy() {  }  } |

noAuth.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <title>Title</title>  </head>  <body>  你没有当前资源的访问权限  </body>  </html> |