day14-Web之过滤器和监听器

1、熟悉什么是过滤器

2、掌握过滤器链

3、掌握过滤器的优先级和参数

4、掌握过滤器的典型应用

5、熟悉什么是监听器

6、掌握常用的监听器

**第十一章 过滤器**

11.1 什么是过滤器

Filter也称之为过滤器，它是Servlet技术中最激动人心的技术，WEB开发人员通过Filter技术，对web服务器管理的所有web资源：例如Jsp, Servlet, 静态图片文件或静态 html 文件等进行拦截，从而实现一些特殊的功能。例如实现URL级别的权限访问控制、过滤敏感词汇、压缩响应信息等一些高级功能。

　　Servlet API中提供了一个Filter接口，开发web应用时，如果编写的Java类实现了这个接口，则把这个java类称之为过滤器Filter。通过Filter技术，开发人员可以实现用户在访问某个目标资源之前，对访问的请求和响应进行拦截

11.2 如何编写过滤器

1、编写java类实现Filter接口

2、重写doFilter方法

3、设置拦截的url

11.3 过滤器的配置

11.3.1 注解式配置

在自定义的Filter类上使用注解@WebFilter(“/\*”)

11.3.2 xml配置

在web.xml中进行过滤器的配置：

|  |
| --- |
| <!--过滤器的xml配置 -->  <filter>  <!--名称-->  <filter-name>sf</filter-name>  <!--过滤器类全称-->  <filter-class>com.qf.web.filter.SecondFilter</filter-class>  </filter>  <!--映射路径配置-->  <filter-mapping>  <!--名称-->  <filter-name>sf</filter-name>  <!--过滤的url匹配规则和Servlet的一模一样-->  <url-pattern>/\*</url-pattern>  </filter-mapping> |

11.4 过滤器链

通常客户端对服务器请求之后，服务器调用Servlet之前会执行一组过滤器（多个过滤器）,那么这组过滤器就称为一条过滤器链。

每个过滤器实现某个特定的功能，一个过滤器检测多个Servlet。（匹配几个，检测几个）。

一组过滤器中的执行顺序与<filter-mapping>的配置顺序呢有关。

当第一个Filter的doFilter方法被调用时，web服务器会创建一个代表Filter链的FilterChain对象传递给该方法。在doFilter方法中，开发人员如果调用了FilterChain对象的doFilter方法，则web服务器会检查FilterChain对象中是否还有filter，如果有，则调用第2个filter，如果没有，则调用目标资源

11.5 过滤器的优先级

在一个web应用中，可以开发编写多个Filter，这些Filter组合起来称之为一个Filter链。web服务器根据Filter在web.xml文件中的注册顺序，决定先调用哪个Filter。当第一个Filter的doFilter方法被调用时，web服务器会创建一个代表Filter链的FilterChain对象传递给该方法。在doFilter方法中，开发人员如果调用了FilterChain对象的doFilter方法，则web服务器会检查FilterChain对象中是否还有filter，如果有，则调用第2个filter，如果没有，则调用目标资源

如果为注解的话，是按照类名的字符串顺序进行起作用的

11.6 过滤器的初始化参数

在过滤器的创建的时候，可以传递初始化参数

第一种：基于注解的

|  |
| --- |
| /\*\*  \* Servlet Filter implementation class FirstFilter 创建过滤器  \*/  @WebFilter(value="/\*",initParams= {@WebInitParam(name = "version", value = "1.0")})  public class FirstFilter implements Filter {  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  throws IOException, ServletException {  // TODO Auto-generated method stub  // place your code here  System.out.println("doFilter……过滤");  // 是否继续---访问下一个  chain.doFilter(request, response);  }  /\*\*  \* @see Filterinit(FilterConfig)  \* 初始化  \*/  public void init(FilterConfig fConfig) throws ServletException {  // TODO Auto-generated method stub  System.out.println("init……初始化");  System.out.println("初始化参数：版本号："+fConfig.getInitParameter("version"));  }  } |

第二种：基于xml配置

|  |
| --- |
| /\*\*  \* 创建过滤器  \*/  public class SecondFilter implements Filter {    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)  throws IOException, ServletException {  // 是否继续---访问下一个  chain.doFilter(request, response);  }  public void init(FilterConfig fConfig) throws ServletException {  // TODO Auto-generated method stub  System.out.println("初始化参数：版本号："+fConfig.getInitParameter("version"));  }  } |

Web.xml实现配置：

|  |
| --- |
| <filter>  <filter-name>myfilter</filter-name>  <filter-class>com.qf.web.filter.SecondFilter</filter-class>  <!--过滤器的初始化参数 -->  <init-param>  <param-name>version</param-name>  <param-value>1.0</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name>myfilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> |

11.7 过滤器的优点

可以实现 Web 应用程序中的预处理和后期处理逻辑

11.8 过滤器的典型应用

11.8.1 禁止浏览器缓存动态页面

过滤器的代码：

|  |
| --- |
| //把ServletRequest强转成HttpServletRequest  HttpServletRequest req = (HttpServletRequest) request;  //把ServletResponse强转成HttpServletResponse  HttpServletResponse resp = (HttpServletResponse) response;  //禁止浏览器缓存所有动态页面  resp.setDateHeader("Expires", -1);  resp.setHeader("Cache-Control", "no-cache");  resp.setHeader("Pragma", "no-cache");  //放行  chain.doFilter(req, resp); |

11.8.2 自动登录

|  |
| --- |
| // 强制转换为Http的请求和响应  HttpServletRequest req = (HttpServletRequest) request;  HttpServletResponse rep = (HttpServletResponse) response;  // 验证是否登录  **if** (req.getSession().getAttribute("user") == **null**) {  // 从Cookie获取上次保存的账号和密码  Cookie[] cks = req.getCookies();  User user = **null**;  **for** (Cookie c : cks) {  **if** (c.getName().equals("user")) {  String[] us = c.getValue().split("@");  user = **new** User();  user.setUsername(us[0]);  user.setPass(us[1]);  **break**;  }  }  // 如果存储Cookie，那么就实现自动登录  **if** (user != **null**) {// 需要自动登录  // 登录校验  User user1 = DbHelper.querySingle("select \* from tb\_user where username=?", User.**class**,  user.getUsername());  **boolean** res = **true**;  **if** (user1 != **null**) {  **if** (user.getPass().equals(user1.getPass())) {  req.getSession().setAttribute("user", user1);  res = **false**;  rep.sendRedirect(req.getServletContext().getContextPath() + "/success.jsp");  }  }  **if** (res) {// 登录失败，之前的记录账号和密码错误  Cookie ck = **new** Cookie("user", "");  ck.setPath("/");  ck.setMaxAge(0);  rep.addCookie(ck);  rep.sendRedirect(req.getServletContext().getContextPath() + "/login.jsp");  }  } **else** {// 直接登录页面  chain.doFilter(request, response);  }  } **else** {// 如果已经登录，那么就直接放行  rep.sendRedirect("success.jsp");  } |

**第十二章 监听器的使用**

12.1 什么是监听器

监听器用于监听web应用中某些对象、信息的创建、销毁、增加，修改，删除等动作的发生，然后作出相应的响应处理。当范围对象的状态发生变化的时候，服务器自动调用监听器对象中的方法。常用于统计在线人数和在线用户，系统加载时进行信息初始化，统计网站的访问量等等

12.2 监听器类型

12.2.1 监听ServletContext的变化

a.监听生命周期

ServletContextListener接口

内部方法：

初始化：contextInitialized

销毁：contextDestroyed

b.监听属性内容变化

ServletContextAttributeListener接口

内部的方法：

attributeAdded：监听属性的添加

attributeRemoved：监听属性的移除

attributeReplaced：监听属性的修改

12.2.2 监听HttpSession变化

a.监听生命周期

HttpSessionListener

内部方法：

sessionCreated：监听Session对象的创建

sessionDestroyed：监听Session对象的销毁

b.监听属性内容变化

HttpSessionAttributeListener

监听HttpSession的内容的变化

内部的方法：

attributeAdded：监听属性的添加

attributeRemoved：监听属性的移除

attributeReplaced：监听属性的修改

c.监听服务器的Session的钝化和活化

HttpSessionActivationListener

监听服务器的钝化和活化

内部方法：

sessionWillPassivate：监听Session内部存储对象的钝化-存储

sessionDidActivate：监听Session内部存储对象的活化---读取

对应类需要实现序列化接口Serializable

d.监听对象的添加和移除

HttpSessionBindingListener

监听对象的添加和移除

内部方法：

valueBound：监听对象的绑定

valueUnbound：监听对象的解除绑定

e.HttpSession的id的变化

这是Servlet3.1新增的

HttpSessionIdListener

监听HttpSession的id的变化

这是Servlet3.1新增的

内部方法：

sessionIdChanged：监听HttpSession的id的变化

12.2.3 监听ServletRequest的变化

a.监听生命周期

ServletRequestListener

监听request对象的初始化和销毁

内部方法：

1、requestInitialized：监听request对象的初始化

2、requestDestroyed：监听request对象的销毁

b.监听属性内容变化

ServletRequestAttributeListener

监听属性内容变化

内部方法：

attributeAdded：监听属性的添加

attributeRemoved：监听属性的移除

attributeReplaced：监听属性的修改

c.监听异步请求

Servelt3.1新增监听器

AsyncListener

监听异步请求

内部方法：

1、onStartAsync：监听异步开始

2、onTimeout：监听超时

3、onError：监听异步的错误信息

4、onComplete：监听异步的完成

12.3 监听器的2种配置

12.3.1 xml的配置

在web.xml中进行配置

|  |
| --- |
| <listener>  <!--直接写出自定义的监听器的类名即可-->  <listener-class>com.qf.web.listener.RequestLeftListener</listener-class>  </listener> |

12.3.2注解式配置

Servlet3.0之后新增的，使用注解@WebListener进行监听器的注册