**Filter and Listener**

**过滤器和监听器**

拓薪教育

刘洋

1. 过滤器 Filter
   1. 初识过滤器
      1. 过滤器概念(熟悉)

过滤器JavaWeb三大组件之一，它与Servlet很相似！过滤器是用来拦截请求的，而不是处理请求的。

当用户请求某个Servlet时，会先执行部署在这个请求上的Filter，如果Filter“放行”，那么会继承执行用户请求的Servlet；如果Filter不“放行”，那么就不会执行用户请求的Servlet。

其实可以这样理解，当用户请求某个Servlet时，Tomcat会去执行注册在这个请求上的Filter，然后是否“放行”由Filter来决定。可以理解为，Filter来决定是否调用Servlet！当执行完成Servlet的代码后，还会执行Filter后面的代码。

单个过滤器示意图:



多个过滤器示意图:



* + 1. 过滤器例子

创建一个类,实现javax.servlet.Filter接口,并实现它的三个方法:

|  |
| --- |
| ***public class MyFilter implements*** Filter**{  @Override  *public void* init(**FilterConfig filterConfig**) *throws ServletException* {  }  @Override  *public void* doFilter(**ServletRequest servletRequest**,** ServletResponse servletResponse**,** FilterChain filterChain**) *throws IOException*, *ServletException* {  *System*.*out*.println("Hello Filter!!!");**  **}  @Override  *public void* destroy() {  } }** |

在web.xml中进行Filter配置:

|  |
| --- |
| **<*filter*>  <*filter-name*>**firstFilter**</*filter-name*>  <*filter-class*>**cn.tx.filter.MyFilter**</*filter-class*> </*filter*> <*filter-mapping*>  <*filter-name*>**firstFilter**</*filter-name*>  <*url-pattern*>**/\***</*url-pattern*> </*filter-mapping*>** |

url-pattern和Servlet的url-pattern的匹配规则一模一样

* 1.完全路径匹配 以/开头 例如 /aaa /aaa/bbb
* 2.目录匹配 以/开头 例如 /aaa/\* /\*
* 3.扩展名匹配 不能以/开头 例如 \*.do \*.action ....

OK了，现在可以尝试去访问index.jsp页面了，看看是什么效果！

当用户访问index.jsp页面时，会执行HelloFilter的doFilter()方法！在我们的示例中，index.jsp页面是不会被执行的，如果想执行index.jsp页面，那么我们需要放行！

|  |
| --- |
| **@Override *public void* doFilter(**ServletRequest servletRequest**,** ServletResponse servletResponse**,** FilterChain filterChain**) *throws IOException*, *ServletException* {  *System*.*out*.println("filter 被执行了!");** filterChain**.doFilter(**servletRequest**,**servletResponse**);  *System*.*out*.println("filter 执行完毕了!"); }** |

有很多同学总是错误的认为，一个请求在给客户端输出之后就算是结束了，这是不对的！其实很多事情都需要在给客户端响应之后才能完成！

* 1. 过滤器详解
     1. 过滤器生命周期(熟悉)

我们已经学习过Servlet的生命周期，那么Filter的生命周期也就没有什么难度了！

* init(FilterConfig)：在服务器启动时会创建Filter实例，并且每个类型的Filter只创建一个实例，从此不再创建！在创建完Filter实例后，会马上调用init()方法完成初始化工作，这个方法只会被执行一次；
* **doFilter(ServletRequest req,ServletResponse res,FilterChain chain)**：这个方法会在用户每次访问“目标资源（<url-pattern>index.jsp</url-pattern>）”时执行，如果需要“放行”，那么需要调用FilterChain的doFilter(ServletRequest,ServletResponse)方法，如果不调用FilterChain的doFilter()方法，那么目标资源将无法执行；
* destroy()：服务器会在创建Filter对象之后，把Filter放到缓存中一直使用，通常不会销毁它。一般会在服务器关闭时销毁Filter对象，在销毁Filter对象之前，服务器会调用Filter对象的destory()方法。
  + 1. FilterConfig(熟悉)

你已经看到了吧，Filter接口中的init()方法的参数类型为FilterConfig类型。它的功能与ServletConfig相似，与web.xml文件中的配置信息对应。下面是FilterConfig的功能介绍：

* ServletContext getServletContext()：获取ServletContext的方法；
* String getFilterName()：获取Filter的配置名称；与<filter-name>元素对应；
* String getInitParameter(String name)：获取Filter的初始化配置，与<init-param>元素对应；
* Enumeration getInitParameterNames()：获取所有初始化参数的名称。

Web.xml配置:

|  |
| --- |
| **<*filter*>  <*filter-name*>**firstFilter**</*filter-name*>  <*filter-class*>**cn.tx.filter.MyFilter**</*filter-class*>  <*init-param*>  <*param-name*>**encode**</*param-name*>  <*param-value*>**UTF-8**</*param-value*>  </*init-param*>  <*init-param*>  <*param-name*>**company**</*param-name*>  <*param-value*>**txjava**</*param-value*>  </*init-param*> </*filter*>** |

Filter读取:

|  |
| --- |
| ***String filterName* =** filterConfig**.getFilterName(); *String company* =** filterConfig**.getInitParameter("company");** Enumeration**<*String*> *names* =** filterConfig**.getInitParameterNames(); *while* (*names*.hasMoreElements()){  *String name* = *names*.nextElement();** String value **=** filterConfig**.**getInitParameter**(**name**); }** |

* + 1. FilterChain(熟悉)

doFilter()方法的参数中有一个类型为FilterChain的参数，它只有一个方法：doFilter(ServletRequest,ServletResponse)。

前面我们说doFilter()方法的放行，让请求流访问目标资源！但这么说不严密，其实调用该方法的意思是，“我（当前Filter）”放行了，但不代表其他人（其他过滤器）也放行。

也就是说，一个目标资源上，可能部署了多个过滤器，就好比在你去北京的路上有多个打劫的匪人（过滤器），而其中第一伙匪人放行了，但不代表第二伙匪人也放行了，所以调用FilterChain类的doFilter()方法表示的是执行下一个过滤器的doFilter()方法，或者是执行目标资源！

如果当前过滤器是最后一个过滤器，那么调用chain.doFilter()方法表示执行目标资源，而不是最后一个过滤器，那么chain.doFilter()表示执行下一个过滤器的doFilter()方法。

* + 1. 过滤器执行顺序(熟悉)

一个目标资源可以指定多个过滤器，过滤器的执行顺序是在web.xml文件中的部署顺序：

|  |
| --- |
| **<*filter*>  <*filter-name*>**filterA**</*filter-name*>  <*filter-class*>**cn.tx.filter.FilterA**</*filter-class*> </*filter*> <*filter-mapping*>  <*filter-name*>**filterA**</*filter-name*>  <*url-pattern*>**/\***</*url-pattern*> </*filter-mapping*>  <*filter*>  <*filter-name*>**filterB**</*filter-name*>  <*filter-class*>**cn.tx.filter.FilterB**</*filter-class*> </*filter*> <*filter-mapping*>  <*filter-name*>**filterB**</*filter-name*>  <*url-pattern*>**/\***</*url-pattern*> </*filter-mapping*>** |

FilterA:

|  |
| --- |
| ***public class FilterA implements*** Filter**{  @Override  *public void* init(**FilterConfig filterConfig**) *throws ServletException* {   }  @Override  *public void* doFilter(**ServletRequest servletRequest**,** ServletResponse servletResponse**,** FilterChain filterChain**) *throws IOException*, *ServletException* {  *System*.*out*.println("filter A 被执行了!");** filterChain**.doFilter(**servletRequest**,**servletResponse**);  *System*.*out*.println("filter A 执行完毕了!");  }  @Override  *public void* destroy() {  } }** |

FilterB:

|  |
| --- |
| ***public class FilterB implements*** Filter**{  @Override  *public void* init(**FilterConfig filterConfig**) *throws ServletException* {   }  @Override  *public void* doFilter(**ServletRequest servletRequest**,** ServletResponse servletResponse**,** FilterChain filterChain**) *throws IOException*, *ServletException* {  *System*.*out*.println("filter B 被执行了!");** filterChain**.doFilter(**servletRequest**,**servletResponse**);  *System*.*out*.println("filter B 执行完毕了!");  }  @Override  *public void* destroy() {  } }** |

执行结果:

|  |
| --- |
| filter A 被执行了!  filter B 被执行了!  servlet 被执行了!!!  filter B 执行完毕了!  filter A 执行完毕了! |

* + 1. 过滤器应用场景(精通)
* 执行目标资源之前做预处理工作，例如设置编码，这种试通常都会放行，只是在目标资源执行之前做一些准备工作；
* 通过条件判断是否放行，例如校验当前用户是否已经登录，或者用户IP是否已经被禁用；
* 在目标资源执行后，做一些后续的特殊处理工作，例如把目标资源输出的数据进行处理；
  + 1. 过滤器设置目标资源(熟悉)

在web.xml文件中部署Filter时，可以通过“\*”来执行目标资源：

|  |
| --- |
| **<*filter-mapping*>  <*filter-name*>**myfilter**</*filter-name*>  <*url-pattern*>**/\***</*url-pattern*> </*filter-mapping*>** |

这一特性与Servlet完全相同！通过这一特性，我们可以在用户访问敏感资源时，执行过滤器，例如：<url-pattern>/admin/\*<url-pattern>，可以把所有管理员才能访问的资源放到/admin路径下，这时可以通过过滤器来校验用户身份。

还可以为<filter-mapping>指定目标资源为某个Servlet，例如：

|  |
| --- |
| **<*servlet*>  <*servlet-name*>**myservlet**</*servlet-name*>  <*servlet-class*>**cn.tx.servlet.TestServlet**</*servlet-class*> </*servlet*> <*servlet-mapping*>  <*servlet-name*>**myservlet**</*servlet-name*>  <*url-pattern*>**/abc**</*url-pattern*> </*servlet-mapping*> <*filter*>  <*filter-name*>**myfilter**</*filter-name*>  <*filter-class*>**cn.tx.filter.MyFilter**</*filter-class*> </*filter*> <*filter-mapping*>  <*filter-name*>**myfilter**</*filter-name*>  <*servlet-name*>**myservlet**</*servlet-name*> </*filter-mapping*>** |

当用户访问http://localhost:8080/filtertest/abc时，会执行名字为myservlet的Servlet，这时会执行过滤器。

* + 1. 过滤器总结
* Filter的三个方法：
* void init(FilterConfig)：在Tomcat启动时被调用；
* void destroy()：在Tomcat关闭时被调用；
* void doFilter(ServletRequest,ServletResponse,FilterChain)：每次有请求时都调用该方法；
* FilterConfig类：与ServletConfig相似，用来获取Filter的初始化参数
* ServletContext getServletContext()：获取ServletContext的方法；
* String getFilterName()：获取Filter的配置名称；
* String getInitParameter(String name)：获取Filter的初始化配置，与<init-param>元素对应；
* Enumeration getInitParameterNames()：获取所有初始化参数的名称。
* FilterChain类：
* void doFilter(ServletRequest,ServletResponse)：放行！表示执行下一个过滤器，或者执行目标资源。可以在调用FilterChain的doFilter()方法的前后添加语句，在FilterChain的doFilter()方法之前的语句会在目标资源执行之前执行，在FilterChain的doFilter()方法之后的语句会在目标资源执行之后执行。
  1. 过滤器应用

过滤器解决POST中文乱码

前台页面:

|  |
| --- |
| **<*form* action="/filter\_demo/test/post" method="post">** 姓名:**<*input* type="text" name="name"/><*br*>  <*input* type="submit" value="提交"> </*form*>** |

Web.xml配置

|  |
| --- |
| **<*filter*>  <*filter-name*>**encodeFilter**</*filter-name*>  <*filter-class*>**cn.tx.filter.EncodeFilter**</*filter-class*>  <*init-param*>  <*param-name*>**encode**</*param-name*>  <*param-value*>**UTF-8**</*param-value*>  </*init-param*> </*filter*> <*filter-mapping*>  <*filter-name*>**encodeFilter**</*filter-name*>  <*url-pattern*>**/\***</*url-pattern*> </*filter-mapping*>** |

Filter代码:

|  |
| --- |
| ***public class EncodeFilter implements*** Filter**{  *private String encode*;  @Override  *public void* init(**FilterConfig filterConfig**) *throws ServletException* {  *this*.*encode* =** filterConfig**.getInitParameter("encode");  }   @Override  *public void* doFilter(**ServletRequest servletRequest**,** ServletResponse servletResponse**,** FilterChain filterChain**) *throws IOException*, *ServletException* {** servletRequest**.setCharacterEncoding(*encode*);** servletResponse**.setContentType("text/html;charSet="+*this*.*encode*);** servletResponse**.setCharacterEncoding(*this*.*encode*);** filterChain**.doFilter(**servletRequest**,**servletResponse**);  }   @Override  *public void* destroy() {   } }** |

Servlet代码:

|  |
| --- |
| ***protected void* doPost(**HttpServletRequest request**,** HttpServletResponse response**) *throws ServletException*, *IOException* {  *String name* =** request**.getParameter("name");  *System*.*out*.println(*name*);** response**.getWriter().write("测试中文乱码"); }** |

1. 监听器 Listener
   1. 监听器概述(了解)

在JavaWeb被监听的事件源为：ServletContext、HttpSession、ServletRequest，即三大域对象。

* 监听域对象“创建”与“销毁”的监听器；
* 监听域对象“操作域属性”的监听器；
* 监听HttpSession的监听器。
  1. ServletContextListener(了解)

ServletContextListener：Tomcat启动和关闭时调用下面两个方法:

* public void contextInitialized(ServletContextEvent evt)：ServletContext对象被创建后调用；
* public void contextDestroyed(ServletContextEvent evt)：ServletContext对象被销毁前调用；
  1. HttpSessionListener(了解)

HttpSessionListener：开始会话和结束会话时调用下面两个方法

* public void sessionCreated(HttpSessionEvent evt)：HttpSession对象被创建后调用；
* public void sessionDestroyed(HttpSessionEvent evt)：HttpSession对象被销毁前调用；
  1. ServletRequestListener

ServletRequestListener：开始请求和结束请求时调用下面两个方法

* public void requestInitiallized(ServletRequestEvent evt)：ServletRequest对象被创建后调用；
* public void requestDestroyed(ServletRequestEvent evt)：ServletRequest对象被销毁前调用。
  1. Listener的使用

Web.xml配置:

|  |
| --- |
| **<*listener*>  <*listener-class*>**cn.tx.listener.MyListener**</*listener-class*> </*listener*>** |

Listener代码:

|  |
| --- |
| ***public class MyListener implements*** ServletContextListener**,** HttpSessionListener**,** ServletRequestListener **{   *public* MyListener() {  }   @Override  *public void* requestInitialized(*ServletRequestEvent*** sre**) {  *System*.*out*.println("requestInitialized");  }   @Override  *public void* requestDestroyed(*ServletRequestEvent*** sre**) {  *System*.*out*.println("requestDestroyed");  }  *public void* contextInitialized(*ServletContextEvent*** sce**) {  *System*.*out*.println("contextInitialized");  }   *public void* contextDestroyed(*ServletContextEvent*** sce**) {  *System*.*out*.println("contextDestroyed");  }  *public void* sessionCreated(*HttpSessionEvent*** se**) {  *System*.*out*.println("sessionCreated");  }   *public void* sessionDestroyed(*HttpSessionEvent*** se**) {  *System*.*out*.println("sessionDestroyed");  }  }** |

* 1. 监听器练习(熟练)

通过HttpSessionListener监听Session的在线和销毁实现在线人员实时统计

index.jsp代码:

|  |
| --- |
| <%@ ***page*** contentType="**text/html;charset=UTF-8**" language="**java**" %> **<*html*>  <*head*>  <*title*>**拓薪教育官网**</*title*>  </*head*>  <*body*>  <*h1*>**欢迎登陆:拓薪学员**</*h1*>  <*h2*>** 当前在线人数:***${count}* </*h2*>  <*hr*>  <*a* href="logout.jsp">**退出**</*a*>  </*body*> </*html*>** |

Logout.jsp代码:

|  |
| --- |
| <%@ ***page*** contentType="**text/html;charset=UTF-8**" language="**java**" %> **<*html*> <*head*>  <*title*>**Title**</*title*> </*head*> <*body*>  *<%*** request**.getSession().invalidate();  *%>* <*h1*>**已退出**</*h1*> </*body*> </*html*>** |

MyListener代码:

|  |
| --- |
| ***public class MyListener implements*** ServletContextListener**,** HttpSessionListener **{** // 定义一个在线人数 初始为0  ***private Integer count* = *0*;  *public* MyListener() {  }  *public void* contextInitialized(*ServletContextEvent*** sce**) {** // 初始化Servlet容器时把在线人数存入  ServletContext ***context* =** sce**.getServletContext();  *context*.setAttribute("count",*count*);  *System*.*out*.println("contextInitialized");  }   *public void* contextDestroyed(*ServletContextEvent*** sce**) {  *System*.*out*.println("contextDestroyed");  }  *public void* sessionCreated(*HttpSessionEvent*** se**) {** // 如果创建了一个session 那么在线人数加1  **onLineCount(**se**.getSession().getServletContext(),*true*);  *System*.*out*.println("sessionCreated");  }   *public void* sessionDestroyed(*HttpSessionEvent*** se**) {** // 如果销毁了一个session 那么在线人数减1  **onLineCount(**se**.getSession().getServletContext(),*false*);  *System*.*out*.println("sessionDestroyed");  }** /\*\*  \* 添加或删除在线人数  \* ***@param* context** \* ***@param* isAdd** \*/  ***private void* onLineCount(**ServletContext context**,*boolean*** isAdd**){  *if*(**isAdd**){  *count* = (*Integer*)**context**.getAttribute("count");** context**.setAttribute("count",++*count*);  }*else*{  *count* = (*Integer*)**context**.getAttribute("count");** context**.setAttribute("count",--*count*);  }  } }** |

1. 学习目标

熟练掌握过滤器的使用及生命周期

掌握过滤器的执行顺序及应用场景

了解监听器的概念及常用监听器

能够使用监听器完善在线人数统计的小例子