Servlet监听器和过滤器

# 监听器

Servlet API中定义了监听器接口，可以用于监听ServletContext,  
HttpSession,ServletRequest对象的生命周期事件。**进行事件驱动编程**。

ServletContextListener 监听Servlet上下文对象初始化或者被销毁

ServletContextAttributeListener 监听Servlet上下文中的属性列表的变化

HttpSessionListener 监听Session生命周期

HttpSessionActionListener 监听session被钝化或者激活

HttpSessionAttributeListener 监听Session属性列表发生的变化

HttpSessionBindingListener 监听Session中是否有对象绑定或者删除，该对象要实现这个接口

ServletRequestListener 监听ServletRequest对象生命周期

ServletRequestAttributeListener 监听ServletRequest属性列表发生的变化

|  |
| --- |
| @WebListener  **public** **class** AppListener **implements** ServletContextListener {  @Override  **public** **void** contextDestroyed(ServletContextEvent sce) {  }  @Override  **public** **void** contextInitialized(ServletContextEvent sce) {  ServletContext servletContext = sce.getServletContext();  Map<String, String> countries = **new** HashMap<String, String>();  countries.put("ca", "Canada");  countries.put("us", "United States");  servletContext.setAttribute("countries", countries);  }  } |

# 过滤器

过滤器：servlet规范当中定义的一种特殊的组件，用来拦截servlet容器的调用过程并进行相应的处理。过滤器中有三个接口Filter, fiterCofnig, Filter  
Chain

## Filter接口

|  |
| --- |
| **init(FilterConfig filterConfig)**：初始化过滤器。filterconfig用来获取ServletContext,初始化参数。容器启动的时候调用。  **doFilter(ServletRequest request,ServletResponse response,FilterChain chain)**：实现过滤功能。可以调用chain.doFilter(request,response)方法将请求传递给下一个过滤器或者目标资源，也可以直接向客户端返回响应信息。  **destory()**：结束过滤器的生命周期。 |

## FilterConfig接口

|  |
| --- |
| getInitParameter(String name)： 返回名为name的初始化参数的值  getInitParameterNames()：返回所有初始化参数的名字的枚举集合。  getServletContext()：返回Servlet上下文对象的引用 |

## FilterChain接口

|  |
| --- |
| doFilter(ServletRequest request,ServletResponse response)：调用该方法将使过滤器链中的下一个过滤器被调用，如果该方法时过滤器链中最后一个过滤器，那么目标资源被调用。 |

## 过滤器的使用

|  |
| --- |
| @WebFilter(filterName = "LoggingFilter", urlPatterns = { "/\*" },  initParams = {  @WebInitParam(name = "logFileName",  value = "log.txt"),  @WebInitParam(name = "prefix", value = "URI: ") })  **public** **class** LoggingFilter **implements** Filter {  **private** PrintWriter logger;  **private** String prefix;  @Override  **public** **void** init(FilterConfig filterConfig)  **throws** ServletException {  prefix = filterConfig.getInitParameter("prefix");  String logFileName = filterConfig  .getInitParameter("logFileName");  String appPath = filterConfig.getServletContext()  .getRealPath("/");  // without path info in logFileName, the log file will be  // created in $TOMCAT\_HOME/bin  System.***out***.println("logFileName:" + logFileName);  **try** {  logger = **new** PrintWriter(**new** File(appPath,  logFileName));  } **catch** (FileNotFoundException e) {  e.printStackTrace();  **throw** **new** ServletException(e.getMessage());  }  }  @Override  **public** **void** destroy() {  System.***out***.println("destroying filter");  **if** (logger != **null**) {  logger.close();  }  }  @Override  **public** **void** doFilter(ServletRequest request,  ServletResponse response, FilterChain filterChain)  **throws** IOException, ServletException {  System.***out***.println("LoggingFilter.doFilter");  HttpServletRequest httpServletRequest =  (HttpServletRequest) request;  logger.println(**new** Date() + " " + prefix  + httpServletRequest.getRequestURI());  logger.flush();  filterChain.doFilter(request, response);  }  }  **#过滤器的优点**  --a,可以实现代码的“可插拔性":即增加或者减少某个功能模块，不会影响程序的  正常执行。  --b可以将多个模块相同的处理逻辑集中写在过滤器里面，方便代码的维护。  <filter>  <filter-name>过滤器名</filter-name>  <filter-class>过滤器类</filter-class>  <init-param>  <param-name>参数名</param-name>  <param-value>参数值</param-value>  </init-param>  </filter>  <filter-mapping>  <filter-name></filter-name>  <url-pattern></url-pattern>  <!--指定过滤器对应的请求方式，默认request-->  <dispatcher>REQUEST,INCLUDE,FORWARD,ERROR</dispatcher>  </filter-mapping>  **注意**：当有多个过滤器都满足过滤条件，则容器按照<filter-mapping>出现的先后顺序来调用各个过滤器。 |