**实验十二 Java Web高级应用**

一、实验目的

1. 了解可以产生事件的对象；

2. 掌握ServletContext和HttpSession对象上的事件处理方法；

3. 了解在ServletRequest对象上的事件及处理方法；

4. 了解过滤器的作用；

5. 掌握过滤器的开发与部署的步骤；

6. 了解过滤器链。

二、实验原理

在Web应用程序中，事件的处理也是通过事件监听器接口处理的。Web应用事件处理的原理为：当Web应用中某些状态改变时，Servlet容器就产生某种事件，如ServletContext对象初始化时会产生ServletContextEvent事件，此时Servlet容器就会从注册的事件监听器中寻找处理该事件的监听器对象，并执行相应的代码。

在Servlet 2.4规范中共定义了6种事件类型和8个事件监听器接口，它们可以处理三种对象上的事件，如表12-1所示：

表12-1 监听器接口与事件类

|  |  |  |
| --- | --- | --- |
| 监听对象 | 监听器接口 | 监听事件 |
| ServletContext | ServletContextListener | ServletContextEvent |
| ServletContextAttributeListener | ServletContextAttributeEvent |
| HttpSession | HttpSessionListener | HttpSessionEvent |
| HttpSessionActivationListener |
| HttpSessionAttributeListener | HttpSessionBindingEvent |
| HttpSessionBindingListener |
| ServletRequest | ServletRequestListener | ServletRequestEvent |
| ServletRequestAttributeListener | ServletRequestAttributeEvent |

过滤器是web服务器上的组件，它们对客户和资源之间的请求和响应进行过滤。

过滤器的工作原理是：当servlet容器接收到对某个资源的请求，它要检查是否有过滤器与之关联。如果有过滤器与该资源关联，servlet容器将把该请求发送给过滤器。在过滤器处理完请求后，它将做下面3件事：

* 产生响应并将其返回给客户；
* 如果有过滤器链，它将把（修改过或没有修改过）请求传递给下一个过滤器；
* 将请求传递给不同的资源。

当请求返回到客户时，它是以相反的方向经过同一组过滤器返回。过滤器链中的每个过滤器够可能修改响应。

过滤器API主要包括：Filter、FilterConfig和FilterChain接口。

三、实验内容与步骤

**1、编写一个ServletContext事件监听器。该事件监听器的功能是当Web应用程序初始化和销毁时以及在ServletContext对象上添加属性、删除属性和替换属性时，在Tomcat日志中记录有关信息。【28‘】**

【步骤1】编写监听器类（10’）

**（1）代码**

package com.listener;

import java.util.Date;

import javax.servlet.\*;

public final class MyServletContextListener implements ServletContextListener, ServletContextAttributeListener {

private ServletContext context = null;

**public void contextInitialized(ServletContextEvent sce) {【2分】**

**context = sce.getServletContext();**

**context.log("应用程序已启动：" + new Date());**

**}**

**public void contextDestroyed(ServletContextEvent sce) {【2分】**

**context = sce.getServletContext();**

**context.log("应用程序已关闭：" + new Date());**

**}**

**public void attributeAdded(ServletContextAttributeEvent sce) {【2分】**

**context = sce.getServletContext();**

**context.log("添加一个属性：" + sce.getName() + "：" + sce.getValue());**

**}**

**public void attributeRemoved(ServletContextAttributeEvent sce) {【2分】**

**context = sce.getServletContext();**

**context.log("删除一个属性：" + sce.getName() + "：" + sce.getValue());**

**}**

**public void attributeReplaced(ServletContextAttributeEvent sce) {【2分】**

**context = sce.getServletContext();**

**context.log("替换一个属性：" + sce.getName() + "：" + sce.getValue());**

**}**

}

【步骤2】完成web.xml，实现注册监听器类。

**（2）代码**（5’）

**<listener>【5分】**

**<listener-class>com.listener.MyServletContextListener</listener-class>**

**</listener>**

【步骤3】完成测试类EventTestServlet.java，实现添加属性（obj=100）、替换属性（obj=当前日期）和删除属性（obj）。

**（3）代码**（10’）

package com.listener;

import java.io.\*;

import java.util.Date;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class EventTestServlet extends HttpServlet {

**ServletContext context = null;【1分】**

**public void init() {【2分】**

**context = getServletConfig().getServletContext();**

**}**

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<html><head><title>Event Test</title></head>");

Integer anInteger = new Integer(100);

Date aDate = new Date();

**context.setAttribute("obj", anInteger); // 添加属性【2分】**

**context.setAttribute("obj", aDate); // 替换属性【2分】**

**context.removeAttribute("obj"); // 删除属性【2分】**

out.println("<h3 align=center>" + "已在ServletContext对象上添加、替换和删除了属性</h3>");

out.println("</body></html>");

out.close();

}

}

【步骤4】完成web.xml，实现注册Servlet。

**（4）代码**（3’）

<servlet>

<servlet-name>EventTestServlet</servlet-name>

<servlet-class>com.listener.EventTestServlet</servlet-class>

</servlet>

**<servlet-mapping>【2分】**

**<servlet-name>EventTestServlet</servlet-name>**

**<url-pattern>/eventtest</url-pattern>**

**</servlet-mapping>**

【步骤5】启动Tomcat，访问EventTestServlet。

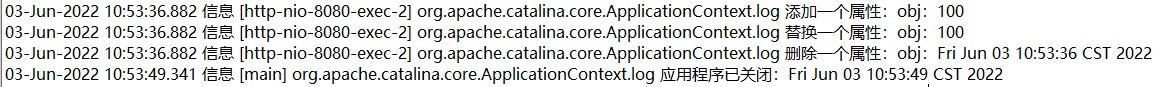


图12-1 EventTestServlet程序的运行结果

**（5）运行截图**

【步骤6】检查日志文件

打开<CATALINA\_HOME>\logs目录中的localhost.XXXXXX.log日志文件，在其中可以找到下面这些行。



上述信息是执行事件监听器后写到日志文件中的。

**（6）运行截图**

**2、编写HttpSession事件监听器用来记录当前在线人数。【28】**

【步骤1】编写HttpSessionListener监听器类

**（1）代码**（10’）

package com.listener;

import javax.servlet.http.HttpSessionEvent;

import javax.servlet.http.HttpSessionListener;

public final class MySessionListener implements HttpSessionListener {

**private int count = 0;【1分】**

**public void sessionCreated(HttpSessionEvent se) {【2分】**

**count++;【1分】**

**se.getSession().getServletContext().setAttribute("onlineCount", new Integer(count));【2分】**

**}**

**public void sessionDestroyed(HttpSessionEvent se) {**

**count--;【1分】**

**se.getSession().getServletContext().setAttribute("onlineCount", new Integer(count));【2分】**

**}**

}

【步骤2】完成web.xml，实现注册事件监听器。

**（2）代码**（5’）

**<listener>【5分】**

**<listener-class>com.listener.MySessionListener</listener-class>**

**</listener>**

【步骤3】编写测试类OnlineCountServlet。

**（3）代码**（10’）

package com.listener;

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class OnlineCountServlet extends HttpServlet {

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

**HttpSession session = request.getSession(true);【2分】**

**ServletContext context = getServletConfig().getServletContext();【2分】**

**Integer count = (Integer) context.getAttribute("onlineCount");【2分】**

response.setContentType("text/html;charset=gb2312");

PrintWriter out = response.getWriter();

out.println("<html><head><title>Session Event Test</title></head>");

**out.println("<h3 align=center><font color=\"#ff0000\">" + "当前在线人数：" + count + "</h3>");【2分】**

out.println("</body></html>");

out.close();

}

}

【步骤4】完成web.xml文件，实现注册Servlet。

**（4）代码**（3’）

<servlet>

<servlet-name>OnlineCountServlet</servlet-name>

<servlet-class>com.listener.OnlineCountServlet</servlet-class>

</servlet>

**<servlet-mapping>【2分】**

**<servlet-name>OnlineCountServlet</servlet-name>**

**<url-pattern>/online</url-pattern>**

**</servlet-mapping>**

【步骤5】访问OnlineCountServlet

再启动两个浏览器窗口访问该Servlet，计数器就会增加，而点击浏览器的“刷新”按钮，计数器不会增加（因为没有开始新会话）。

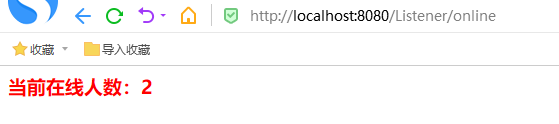


图12-2 OnlineCountServlet程序的运行结果

1. **运行截图**

**3、编写过滤器审计用户对资源的访问。【14】**

【步骤1】完成AuditFilter.java过滤器，实现当用户访问应用程序任何资源时，将用户的IP地址和主机名写入日志文件中。

**（1）代码**（10’）

package com.filter;

import java.io.IOException;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class AuditFilter implements Filter {

**protected FilterConfig config;【2分】**

public void init(FilterConfig filterConfig) throws ServletException {

this.config = filterConfig;

}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

throws IOException, ServletException {

**HttpServletRequest req = (HttpServletRequest) request;【2分】**

**HttpServletResponse res = (HttpServletResponse) response;【2分】**

**String addr = req.getRemoteAddr();【2分】**

**String user = req.getRemoteHost();【2分】**

**config.getServletContext().log("RemoteAddress:" + addr + ",RemoteHost:" + user);【2分】**

**chain.doFilter(req, res);【2分】**

}

public void destroy() {

}

}

【步骤2】在部署描述文件中配置过滤器，实现使用URL模式为/\*，将该过滤器映射到Web应用程序的任何资源。

**（2）代码**（4’）

**<filter>【2分】**

**<filter-name>AuditFilter</filter-name>**

**<filter-class>com.filter.AuditFilter</filter-class>**

**</filter>**

**<filter-mapping>【2分】**

**<filter-name>AuditFilter</filter-name>**

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

**</filter-mapping>**

【步骤3】创建onlineCount.jsp后进行访问。

打开<CATALINA\_HOME>\logs目录中的localhost.XXXX.log文件中有下面一行（访问多个资源就会有多行）：



**（3）运行效果**

**4、编写过滤器实现改变请求编码。（30）**

【步骤1】完成loginform.html，实现利用表单获取用户名和密码，将用户输入内容提交至由servlet/check映射的Servlet处理。



图12-3 loginform.html程序的运行结果

**（1）代码**（5’）

<html>

<head>

<title>使用过滤器改变请求编码</title>

<meta http-equiv="Content-Type" content="text/html;charset=GB2312">

</head>

<body>

<h2>请输入用户名和口令：</h2>

**<form method="post" action="servlet/check">【2分】**

<table>

<tr>

<td>用户名：</td>

<td><input name="name" type="text"></td>

</tr>

<tr>

<td>口 令：</td>

<td><input name="pass" type="password"></td>

</tr>

<tr>

<td></td>

<td>

<input name="ok" type="submit" value="提交">

<input name="cancel" type="reset" value="重置">

</td>

</tr>

</table>

</form>

</body>

</html>

【步骤2】完成CheckParamServlet.java，实现处理请求参数，并响应至前端显示用户名和口令。

**（2）代码**（5’）

package com.filter;

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class CheckParamServlet extends HttpServlet {

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

**String name = request.getParameter("name");【2分】**

**String pass = request.getParameter("pass");【2分】**

response.setContentType("text/html;charset=gb2312");

PrintWriter out = response.getWriter();

out.println("<html><head><title>Param Test</title></head>");

out.println("<h3 align=center>你的用户名为：" + name + "</h3>");

out.println("<h3 align=center>你的口令为：" + pass + "</h3>");

out.println("</body></html>");

}

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

doGet(request, response);

}

}

【步骤3】完成web.xml，实现注册Servlet。

**（3）代码**（5’）

**<servlet>【2分】**

**<servlet-name>CheckParamServlet</servlet-name>**

**<servlet-class>com.filter.CheckParamServlet</servlet-class>**

**</servlet>**

**<servlet-mapping>【2分】**

**<servlet-name>CheckParamServlet</servlet-name>**

**<url-pattern>/servlet/check</url-pattern>**

**</servlet-mapping>**

【步骤4】访问loginform.html。

**（4）运行截图**

点击“提交”按钮，经CheckParamServlet处理后返回的结果如下图所示：

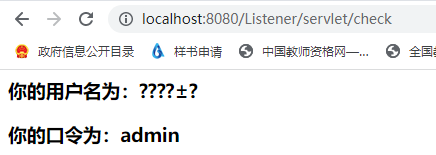


图12-4 CheckParamServlet程序的运行结果

可以看到，从服务器返回的汉字成了乱码。原因是没有指定request的编码。

接下来通过编写过滤器改变请求编码。

【步骤5】完成EncodingFilter.java，实现过滤器改变请求编码。

**（5）代码**（10’）

package com.filter;

import java.io.IOException;

import javax.servlet.\*;

public class EncodingFilter implements Filter {

protected String encoding = null;

**protected FilterConfig config;【2分】**

public void init(FilterConfig filterConfig) throws ServletException {

this.config = filterConfig;

// 得到在web.xml中配置的编码

**this.encoding = filterConfig.getInitParameter("Encoding");【2分】**

}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

throws IOException, ServletException {

if (request.getCharacterEncoding() == null) {

// 得到指定的编码

**String encode = getEncoding();【2分】**

if (encode != null) {

// 设置request的编码

**request.setCharacterEncoding(encode);【2分】**

**response.setCharacterEncoding(encode);【2分】**

}

}

**chain.doFilter(request, response);【2分】**

}

protected String getEncoding() {

return encoding;

}

public void destroy() {

}

}

【步骤6】完成web.xml，实现配置过滤器。

**（6）代码**（5）

**<filter>【3分】**

**<filter-name>EncodingFilter</filter-name>**

**<filter-class>com.filter.EncodingFilter</filter-class>**

**<init-param>**

**<param-name>encoding</param-name>**

**<param-value>gb2312</param-value>**

**</init-param>**

**</filter>**

**<filter-mapping>【2分】**

**<filter-name>EncodingFilter</filter-name>**

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

**</filter-mapping>**

【步骤7】重复第（4）步操作，访问loginForm.html。



图12-5 EncodingFilter程序的运行结果

**（7）运行截图**