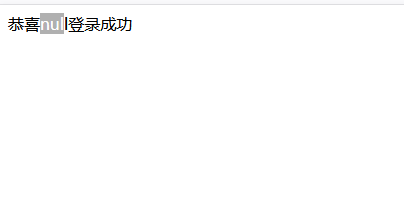
<%@ **page** language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<html>  
<body>  
<%  
 String username=(String) session.getAttribute("user");  
%>  
恭喜<%=username %>登录成功  
</body>  
</html>



JSP登录成功页面分析与实验心得

核心内容分析

1.Session 数据获取

通过 session.getAttribute("user") 从会话中提取用户名（需先由 Servlet 或登录页存入）。

强制类型转换：(String) 确保数据类型匹配。

2.动态内容渲染

使用 JSP 表达式 <%=username %> 将 Java 变量嵌入 HTML，实现个性化消息：

恭喜xxx登录成功。

3.字符编码一致性

charset=UTF-8 确保中文字符正常显示（如"恭喜"、"登录成功"）。

实验心得：登录验证过滤器实现

1. 过滤器核心作用

安全拦截：在请求到达 JSP/Servlet 前验证用户登录状态。

统一控制：避免在每个页面重复编写验证逻辑。

2. 关键实现步骤

java

// LoginFilter.java

public class LoginFilter implements Filter {

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

HttpServletRequest request = (HttpServletRequest) req;

HttpServletResponse response = (HttpServletResponse) res;

// 1. 检查 session 中是否存在用户标识

HttpSession session = request.getSession(false);

if (session == null || session.getAttribute("user") == null) {

// 2. 未登录时重定向到登录页

response.sendRedirect(request.getContextPath() + "/login.jsp");

return;

}

// 3. 已登录则放行请求

chain.doFilter(request, response);

}

}

3. web.xml 配置示例

xml

<filter>

<filter-name>LoginFilter</filter-name>

<filter-class>com.example.LoginFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>LoginFilter</filter-name>

<!-- 保护所有需要登录的页面 -->

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

<!-- 排除登录页本身 -->

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

<dispatcher>REQUEST</dispatcher>

</filter-mapping>

总结：该 JSP 页面是会话验证的终点展示，需结合过滤器和 Servlet 实现完整安全闭环。过滤器作为"守门人"确保非法用户无法访问受限资源，而 JSP 则友好展示已验证用户的信息。