JSP 动态网页技术(Vue 逐渐取代这个技术)

课程介绍：
在 WEB 项目开发中 JSP 占据着举足轻重的地位，虽然其使用很简单，但是非常的重要。本节课开始学习 JSP。

JSP 学习:

问题：
在学习了 Servlet 之后，使用 Servlet 进行页面的展现，代码书写过于麻烦。极大的影响了开发的效率，那么有没有一种方式可以让我们像以前写网页一样来进行网页的编程工作呢?

解决：
使用 JSP 技术

概念：
JSP 全名为 Java Server Pages，中文名叫 java 服务器页面，其根本是一个简化的 Servlet 设计，它 [1] 是由 Sun Microsystems 公司倡导、许多公司参与一起建立的一种动态网页技术标准。

什么是动态网页？

JSP（全称 JavaServer Pages）是由 Sun Microsystems 公司主导创建的一种动态网页技术标准。JSP 部署于网络服务器上，可以响应客户端发送的请求，并根据请求内容动态地生成 HTML、XML 或其他格式文档的 Web 网页，然后返回给请求者。JSP 技术以 Java 语言作为脚本语言，为用户的 HTTP 请求提供服务，并能与服务器上的其它 Java 程序共同处理复杂的业务需求。

特点：

本质上还是 Servlet
跨平台，一次编写处处运行***.jsp---->***.class
组件跨平台
健壮性和安全性

Jsp 的访问原理：

浏览器发起请求，请求 JSP，请求被 Tomcat 服务器接收，执行

JspServlet 将请求的 JSP 文件转义成为对应的 java 文件 (也是

Servlet)，然后执行转义好的 java 文件。

第一个 jsp，并解析原理：
实际运行是
C:\Users\Administrator\.IntelliJIdea2019.3\system\tomcat 文件原理如下:

```
<servlet>
    <servlet-name>jsp</servlet-name>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
    <init-param>
      <param-name>fork</param-name>
      <param-value>false</param-value>
    </init-param>
    <init-param>
      <param-name>xpoweredBy</param-name>
      <param-value>false</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
  </servlet>

<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
    <url-pattern>*.jspx</url-pattern>
  </servlet-mapping>
```

public class JspServlet extends HttpServlet

这个 JspServlet 是可以把 jsp 文件转义成 serlvet 文件 aa_jsp 的,最终实际在 tomcat 中运行的是 aa_jsp 的 servlet 文件

所以,从始至终 tomcat 服务器根本就没有见过 jsp 这个文件

Jsp 的语法和指令：

page 的作用：

配置 jsp 文件的转译相关的参数。

JSP 的 page 指令介绍：

```
<%@ page language="java"  pageEncoding="UTF-8" import="java.io.*,java.awt.*"
session="true" errorPage="error.jsp" %>
<%-- JSP 中的注释
<%@   %> :包括 jsp 文件相关描述信息的标签,可以含有各种属性
page 属性:描述当前 jsp 文件
contentType:当前文件响应编码等同于 servlet 中 resp.setContentType
language:表示当前 jsp 是用 java 语言作为脚本
pageEncoding:表示当前 jsp 文件的编码,同时也是设置响应编码,与 contentType 有相同的效果,所以可以
只写 pageEncoding
import="com.io.*" :用来导入 java 类的,如果有多个用逗号隔开
session:设置当前 jsp 页面是否支持 session,默认不支持,session="true"开启 session 的支持
```

```
errorPage:如果当前 jsp 页面有错,就会跳转到 errorPage 指定的页面
--%>
    <html>
    <head>
        <title>Title</title>
</head>
<body>
    这是我的第一个 jsp 页面
    <%
        int a = 10/0;
    %>
</body>
</html>
```

Jsp 的三种注释

前端语言注释：

会被转译，也会被发送，但是不会被浏览器执行

java 语言注释：

会被转译，但是不会被 servlet 执行

Jsp 注释：

不会被转译。

script 注释

按各自的注释来写,相互不影响

Jsp 的 page 指令学习：

<%@page 属性名="属性值" 属性名="属性值"...%>

Jsp 的注释 <%-- --%>

Jsp 的局部代码块

特点：

局部代码块中声明的 java 代码会被原样转译到 jsp 对应的 servlet 文件的_JspService 方法中

代码块中声明的变量都是局部变量。

使用：<% java 代码 %>

缺点：

使用局部代码块在 jsp 中进行逻辑判断，书写麻烦，阅读困难。

开发：

servlet 进行请求逻辑处理，使用 jsp 进行页面展现。

Jsp 的全局代码块

特点：

声明的 java 代码作为全局代码转译到对应的 servlet 类中。

使用：

<%! 全局代码 %>

注意：

全局代码块声明的代码，需要使用局部代码块调用。

Jsp 的脚本段语句：

特点：帮助我们快速的获取变量或者方法的返回值作为数据响应给浏览器。

使用：<%=变量名或者方法%> 等同于 resp.getWriter().writer()

注意：不要在变量名或者方法后使用分号。

位置：除 jsp 语法要求以外的任意位置。

Jsp 的脚本端<%= 变量名或方法 %>

```
<%@ page language="java"  pageEncoding="UTF-8" import="java.io.*,java.awt.*"
session="true" errorPage="error.jsp" %>
<%@ page import="oracle.jdbc.internal.XSCacheOutput" %>
<%@ page import="org.w3c.dom.ls.LSOutput" %>
<%-- JSP 中的注释
<%@   %> :包括 jsp 文件相关描述信息的标签,可以含有各种属性
page 属性:描述当前 jsp 文件
contentType:当前文件响应编码等同于 servlet 中 resp.setContentType
language:表示当前 jsp 是用 java 语言作为脚本
pageEncoding:表示当前 jsp 文件的编码,同时也是设置响应编码,与 contentType 有相同的想过,所以可以
只写 pageEncoding
import="com.io.*" :用来导入 java 类的,如果有多个用逗号隔开
session:设置当前 jsp 页面是否支持 session,默认不支持,session="true"开启 session 的支持
errorPage:如果当前 jsp 页面有错,就会跳转到 errorPage 指定的页面
--%>
    <html>
    <head>
        <title>Title</title>
</head>
<body>

   这是我的第一个 jsp 页面
    <%  //这个标签是局部代码标签
        /*写 java 代码是局部变量,*/
        int a = 10;
        int b= 20;
        System.out.println(a+b);
        System.out.println("c 的值"+c);
        System.out.println(add(a,b));
    %>
   a 的值:
   <%= a%><br/>
   b 的值:
   <%= b%> <br/>
   计算结果是:
   <%= add(23,20) %>   <%--这里等同于 servlet 中写的
resp.getWriter().writer(add(23,20));--%>
```

```
<%!  //中写的代码是成员变量,称为成员代码块标签
    int c = 30;
    //请求的方法
    public int add(int item1,int item2){
       return (item1+item2);
    }
%>
</body>
</html>
```

Jsp 的静态引入和动态引入

Jsp 的静态引入:

<%@include file="要引入的 jsp 文件的相对路径" %>

特点：

会将引入的 jsp 文件和当前 jsp 文件转译成一个 java(Servlet)文件使用。

在网页中也就显示了合并后的显示效果。

注意：

静态引入的 jsp 文件不会单独转译成 java(Servlet)文件。

当前文件和静态引入的 jsp 文件中不能够使用 java 代码块声明同名变量。

Jsp 的动态引入：

<jsp:include page="要引入的 jsp 文件的相对路径"></jsp:include>

特点：

会将引入的 jsp 文件单独转译，在当前文件转译好的 java 文件中调用引入的 jsp 文件的转译文件。

在网页中显示合并后的显示效果。

注意：

动态引入允许文件中声明同名变量。

优点：

降低 jsp 代码的冗余，便于维护升级

```
<%@ page language="java"  pageEncoding="UTF-8" import="java.io.*,java.awt.*"
session="true" errorPage="error.jsp" %>
<%@ page import="oracle.jdbc.internal.XSCacheOutput" %>
<%@ page import="org.w3c.dom.ls.LSOutput" %>

    <html>
    <head>
        <title>Title</title>
</head>
<body>

   这是我的第一个 jsp 页面
<%!
    int a =10;
%>
```

```
<%--静态引入,会把bb.jsp 文件的中的内容融入到aa.jsp 文件中,变为一个整体,注意申明java 块中的java 代码不能使用相同的变量名--%>
<%@include file="bb.jsp"%>
</body>
</html>
```

```
<%@ page language="java"  pageEncoding="UTF-8" import="java.io.*,java.awt.*"
session="true" errorPage="error.jsp" %>
<%@ page import="oracle.jdbc.internal.XSCacheOutput" %>
<%@ page import="org.w3c.dom.ls.LSOutput" %>

    <html>
    <head>
        <title>Title</title>
</head>
<body>

   这是我的第一个 jsp 页面
<%
    int a =10;
%>
   <%--动态引入页面,可以使用相同的java 代码的变量名,因为他们被转译成两个独立的java 文件--%>
<jsp:include page="bb.jsp"></jsp:include>
</body>
</html>
```

JSP 的 taglib 指令(了解)

页面转发(forword 标签)

jsp 的转发标签 forward：

使用：

<jsp:forward page="要转发的 jsp 文件的相对路径"></jsp:forward>

特点：

一次请求

地址栏信息不改变。

注意：

在转发标签的两个标签中间除了写<jsp:param name="str" value="aaa" />子标签不会报错，其他任意字符都会报错。

<jsp:param name="str" value="aaa" />

name 属性为附带的数据的键名　(相当于 intput 标签中的 name)

value 为附带的数据内容　　(相当于 intput 标签中的 value)

```
<%@ page language="java"  pageEncoding="UTF-8" import="java.io.*,java.awt.*"
session="true" errorPage="error.jsp" %>
```

```jsp
<%@ page import="oracle.jdbc.internal.XSCacheOutput" %>
<%@ page import="org.w3c.dom.ls.LSOutput" %>

    <html>
    <head>
        <title>Title</title>
</head>
<body>
   <%
       request.setCharacterEncoding("utf-8");
   %>
    <%--jsp 中页面的转发,类似于 serlvet 中请求转发,<jsp:forward></jsp:forward>标签之间只能写<jsp:param name="" value=""/>,其他信息都不让写--%>
    <jsp:forward page="ServletBB">
        <jsp:param name="uname" value="张安"/>
    </jsp:forward>
</body>
</html>
```

```java
package com.servlet;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "ServletBB",urlPatterns = "/ServletBB")
public class ServletBB extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        req.setCharacterEncoding("utf-8");
        resp.setContentType("text/html;charset=utf-8");
        String uname = req.getParameter("uname");
        resp.getWriter().println("取到从 aa.jsp 中传过来的:"+uname);
    }
}
```

了解,开发中用的不多

开发中应用场景:

servlet: 用来处理逻辑(跳转，java 请求处理)

jsp: 用来展示页面

Jsp 的内置对象

jsp 的九大内置对象:

内置对象：

jsp 文件在转译成其对应的 Servlet 文件的时候自动生成的并声明的对象。我们在 jsp 页面中直接使用即可。

注意：

内置对象在 jsp 页面中使用，使用局部代码块或者脚本段语句来使用。不能够在全局代码块中使用。

内容：九个对象

pageContext:页面上下文对象，封存了其他内置对象。封存了当前 jsp 的运行信息。

注意：每个 Jsp 文件单独拥有一个 pageContext 对象。

作用域：当前页面。

request：封存当前请求数据的对象。由 tomcat 服务器创建。一次请求

session:此对象用来存储用户的不同请求的共享数据的。一次会话

application：也就是 ServletContext 对象，一个项目只有一个。存储用户共享数据的对象，以及完成其他操作。项目内

response:响应对象，用来响应请求处理结果给浏览器的对象。设置响应头，重定向。

out:响应对象，Jsp 内部使用。带有缓冲区的响应对象，效率高于 response.getWriter()对象。(等同于 resp.getWriter())

page:代表当前 Jsp 的对象。相当于 java 中的 this。

exception：异常对象。存储了当前运行的异常信息。

注意：使用此对象需要在 page 指定中使用属性 isErrorPage="true"开启。

config：也就是 ServletConfig，主要是用来获取 web.xml 中的配置数据，完成一些初始化数据的读取。

四个作用域对象:

pageContext:当前页面.解决了在当前页面内的数据共享问题。获取其他内置对象。

request:一次请求。一次请求的 servlet 的数据共享。通过请求转发，将数据流转给下一个 servlet。

session:一次会话.一个用户的不同请求的数据共享。将数据从一次请求流转给其他请求。

application:(等同于 ServletContext)项目内.不同用户的数据共享问题。将数据从一个用户流转给其他用户。

Out-->resp.getWriter()　是个打印流对象　resp 是 HttpServletReponse 对象

resp.setHeader()

作用：

数据流转

El 表达式和 Jstl 标签库学习

El 表达式学习

1、　什么是 EL 表达式

全称：Expression Language，一种写法非常简洁的表达式。语法简单易懂，便于使用。

表达式语言的灵感来自于 ECMAScript 和 XPath 表达式语言

2、 EL 表达式的作用

作用：让 jsp 书写起来更加的方便。简化在 jsp 中获取作用域或者请求数据的写法。也会搭配 Jstl 来进行使用。

3、 使用 EL 表达式

语法结构：${expression},提供.和[ ]两种运算符来存取数据。

使用 El 表达式获取作用域数据:

作用：获取作用域对象中的数据。

注意：获取的是 pageContext、request、session、application 四个对象中的数据，其他数据一概不理会。找到了则获取返回，找不到则什么都不做，也不报错。

●EL 表达式

> 是为了简化咱们的 jsp 代码，具体一点就是为了简化在 jsp 里面写的那些 java 代码。

* 写法格式: ${表达式 }

```
<%@ page import="com.web.pojo.T_User" %>
<%@ page import="java.util.ArrayList" %>
<%@ page import="java.util.HashMap" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    String path = request.getContextPath();
    String basePath = request.getScheme() + "://" + request.getServerName() + ":" +
request.getServerPort() + path + "/";
%>
<html>
<head>
    <base href="<%=basePath%>">
    <title>Title</title>
</head>
<body>
    <h3>el 表达式</h3>
    <%--
    el 表达式:
        用来取作用域中的数据
        session/request/servletcontext/pageContext
    --%>
    <%
        T_User tuser = new T_User();
        tuser.setUname("小美");
        tuser.setPwd("111");
        session.setAttribute("tuser",tuser);
        session.setAttribute("city", "北京");
        session.setAttribute("uname", tuser.getUname());
        ArrayList<String> list = new ArrayList<>();
```

```
        list.add("上海");
        list.add("南京");
        list.add("广州");
        session.setAttribute("list", list);
        HashMap<String,Integer> map = new HashMap<>();
        map.put("a", 10);
        map.put("b", 20);
        map.put("c", 30);
        session.setAttribute("m", map);
        ArrayList<T_User> users = new ArrayList<>();
        users.add(new T_User("张三", "123") );
        users.add(new T_User("李四", "321") );
        users.add(new T_User("王五", "456") );
        session.setAttribute("users", users);
    %>
    <h4>通过 el 表达式取值</h4>
    <%--根据键取值,如果键是对象,那么可以:键.属性取值--%>
    ${tuser.uname}--${tuser.pwd}<br/>
    ${city}--${uname}<br/>
    ${list.get(0)}--${list.get(1)}--${list.get(2)}<br/>
    ${m.a} -- ${m.b}-- ${m.c}<br/>
    ${users.get(0).uname}--${users.get(0).pwd}<br/>
    ${users.get(1).uname}--${users.get(1).pwd}<br/>
    ${users.get(2).uname}--${users.get(2).pwd}<br/>
    <%--无论是 session/request/servletcontext/pageContext 都可以使用以上的方式取各作用域中的数据--%>
<hr/>
<%
    /*四种常用的作用域放入同一个键,不同的值*/
    //session.setAttribute("a", "session");
    //request.setAttribute("a", "request");
    application.setAttribute("a", "application");
    //pageContext.setAttribute("a", "pageContext");
%>
<h4>测试作用域的取值优先顺序</h4>
    ${a}
    <br/>
    <b>
        优先级: pageContext > request > session > application
    </b>
</body>
</html>
```