第十一章：JSP

第1节：简介

> JSP（全称JavaServer Pages）java服务器页面。 是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，是简化版的Servlet。
>
> 执行的过程为 xxx.jsp---->xxx.java---->xxx.class

第2节：使用策略

> 1.纯碎的HTML使用
> 语法和使用HTML的用法完全一样,可以包含css,javaScript,Jquery等
>
> 2.纯粹的Servlet使用
> 以<% java代码片段 %>写法显示在jsp页面，相当于Servlet中的service方法 如果是Java代码写成什么样显示什么样
>
> 3.HTML + Servlet使用
> 如果写的html代码 相当于 out.write("<a>")
> 但是我们需要注意：
> jsp 只用来显示数据 不处理业务
> Servlet 只处理业务

第3节：核心内容

3.1 三大指令

3.1.1 page指令

> ```
> <%@page 属性名="值" 属性名="值" %> 控制整个jsp页面的配置信息
> language="java" 支持的语言
> contentType="text/html; charset=UTF-8" 内容类型
> pageEncoding="UTF-8" 页面的编码格式
> import="java.util.List,java.util.ArrayList" 导入jar
> autoFlush="true" 自动刷新
> errorPage="03-error.jsp" 如果当前页面抛异常 跳转指定页面
> session="true" 当前页面中可以直接使用该对象
> isELIgnored="false" 是否忽略EL表达式
> ```

3.1.2 include指令

> 页面包含说的是一个页面可以包含另一个页面的内容，并且能正常加载。
> 语法：
> `<%@include file=”被包含的页面名称” %>`
> 作用：
> include指令的主要作用：可以将页面公共的部分提取出来，让每个页面包含这个公共的页面，一是为了减少代码量，二是易于手续的维护。

3.1.3 taglib指令

> 后面使用JSTL标准标签库时使用。
> 语法：`<%@taglib prefix="前缀" uri="引入外部标签库的路径" %>`

3.2 六大动作

> 作用：
> JSP动作标签利用XML语法格式的标记来控制Servlet引擎的行为。利用JSP动作可以动态地插入文件、重用JavaBean组件、把用户重定向到另外的页面、为Java插件生成HTML代码。

> 演示：

```
1. <jsp:forward ></jsp:forward> 完全转发
   <jsp:forward page="01-login.jsp"></jsp:forward>
2. <jsp:include ></jsp:include> 部分转发 也成为动态包含
   <jsp:include page="01-login.jsp"></jsp:include>
3. <jsp:useBean></jsp:useBean>
  <%-- 3. 创建实体类对象 --%>
    <jsp:useBean id="users01" class="com.ujiuye.bean.Users"></jsp:useBean>
4. <jsp:setProperty />
   <%-- 4. 实体类对象中赋值 --%>
   <jsp:setProperty property="username" name="users01" value="王二小"/>
5. <jsp:getProperty />
   <%-- 5. 获取对象中的属性值 --%>
  <jsp:getProperty property="username" name="users01"/>
```

面试题：动态包含和静态包含有什么区别?

1、写法不同：

静态包含是指令包含<%@include file="***"%>

动态包含是JSP动作标签包含<jsp:include page="***"></jsp:include>

2、编译方式不同：

静态包含：编译过程中将被包含的页面先整合到包含页面再编译。所以只有一个文件

动态包含：编译过程中包含的页面与被包含的页面都编译，运行后内容再通过servlet整合

3、参数传递：

静态包含不能向被包含页面传递参数

动态包含可以使用jsp:param标签向被包含页面传递参数

4、运行效率：

静态包含快，当成了一个页面；动态包含是多个页面，运行慢。

3.3 九大内置对象

内置对象：也称隐式对象，在jsp中，已经初始化好并且给我们封装好了，不需要程序员创建，可以直接使用的对象就是内置对象。

对象名	功能	类型
Out	页面输出对象	JspWriter
Exception	异常处理对象	Throwable
Page	描述页面的对象	Object
PageContext	页面的全局对象【作用域】	PageContext
Request	请求对象【作用域】	HttpServletRequest
Response	响应对象	HttpServletResponse
Session	会话对象【作用域】	HttpSession
Config	配置	ServletConfig
Application	服务器对象【作用域】	ServletContext

这些内置对象在Jsp页面都是存在的，我们可以直接使用；

这里默认能显示的内置对象共8个，还有一个特殊的对象Exception关于异常的，只有在page指令中设置isErrorPage="true"时才会显示。

可以通过以下方式获取：

```
pageContext.getException() //获取异常对象
```

3.3.1 对象详解

1.application 对象可将信息保存在服务器中，直到服务器关闭，否则application对象中保存的信息会在整个应用中都有效。该对象的类型是SevletContext类型，它的生命周期在tomcat服务器启动时创建，关闭时销毁。其作用范围为整个Web应用。

2.request 对象是 javax.servlet.HttpServletRequest类型的对象。该对象代表了客户端的请求信息，主要用于接受通过HTTP协议传送到服务器的数据。request对象的作用域为一次请求。

3.response 对象是 javax.servlet.HttpServletRsponse类型的对象。该对象代表客户端的响应信息，主要是将JSP容器处理过的对象信息返回到客户端。

4.config 对象的主要作用是取得服务器的配置信息。通过 pageConext对象的 getServletConfig() 方法可以获取一个config对象。当一个Servlet 初始化时，容器把某些信息通过 config对象传递给这个 Servlet。 开发者可以在web.xml 文件中为应用程序环境中的Servlet程序和JSP页面提供初始化参数。

5.page：当前jsp页面的实例，相当于this关键字。

6.session：会话对象。session 对象是由服务器自动创建与用户请求相关的对象。服务器为每个用户都生成一个session对象，用于保存该用户的信息，跟踪用户的操作状态。其作用范围为一次会话。

7.exception对象的作用是显示异常信息，只有在包含 isErrorPage="true" 的页面中才可以被使用，在一般的JSP页面中使用该对象将无法编译JSP文件。excepation对象和Java的所有对象一样，都具有系统提供的继承结构。exception 对象几乎定义了所有异常情况。在Java程序中，可以使用try/catch关键字来处理异常情况； 如果在JSP页面中出现没有捕获到的异常，就会生成 exception 对象，并把 exception 对象传送到在page指令中设定的错误页面中，然后在错误页面中处理相应的 exception 对象。

8.out 对象用于在Web浏览器内输出信息，并且管理应用服务器上的输出缓冲区。JSP中的out对象是JspWriter类型。

9.pageContext 对象的作用是取得任何范围的参数，通过它可以获取 JSP页面的out、request、reponse、session、application 等对象。pageContext对象的创建和初始化都是由容器来完成的，可以获取项目的根目录，还可通过getXXX（）获取其他内置对象

3.3.2 作用域

1. 作用

为了在页面、请求、和用户之间传递和共享数据，JSP提供了四个不同的作用域：pageContext（页面作用域）、request（请求作用域）、session（会话作用域）、application（应用程序作用域），这些作用域就规定了数据可以传递和共享的范围以及数据的存活时间。

2. 详解

1.application 作用域
　如果把变量放到application里，就说明它的作用域是application，它的有效范围是整个应用。 整个应用是指从应用启动，到应用结束。我们没有说“从服务器启动，到服务器关闭”，是因为一个服务器可能部署多个应用，当然你关闭了服务器，就会把上面所有的应用都关闭了。 application作用域里的变量，它们的存活时间是最长的，如果不进行手工删除，它们就一直可以使用。

application作用域上的信息传递是通过ServletContext实现的，它提供的主要方法如下所示：

```
Object getAttribute (String name)   //从application中获取信息;
void setAttribute (String name, Object value)   //向application作用域中设置信息。
```

2.session作用域

session作用域比较容易理解，同一浏览器对服务器进行多次访问，在这多次访问之间传递信息，就是session作用域的体现。如果把变量放到session里，就说明它的作用域是session，它的有效范围是当前会话。所谓当前会话，就是指从用户打开浏览器开始，到用户关闭浏览器这中间的过程。这个过程可能包含多个请求响应。也就是说，只要用户不关浏览器，服务器就有办法知道这些请求是一个人发起的，整个过程被称为一个会话（session），而放到会话中的变量，就可以在当前会话的所有请求里使用。
session是通过HttpSession接口实现的，它提供的主要方法如下所示：

```
Object HttpSession.getAttribute (String name)    //从session中获取信息。
void HttpSession.setAttribute (String name, Object value) //向session中保存信息。
HttpSession HttpServletRequest.getSessio()  //获取当前请求所在的session的对象。
```

session的开始时刻比较容易判断，它从浏览器发出第一个HTTP请求即可认为会话开始。但结束时刻就不好判断了，因为浏览器关闭时并不会通知服务器，所以只能通过如下这种方法判断：如果一定的时间内客户端没有反应，则认为会话结束。Tomcat的默认值为30分钟，但这个值也可以通过HttpSession的setMaxInactiveInterval()方法来设置：

```
void setMaxInactiveInterval(int interval)
```

如果想主动让会话结束，例如用户单击“注销”按钮的时候，可以使用 HttpSession 的 invalidate()方法，用于强制结束当前session：void invalidate()

3.request作用域

一个HTTP请求的处理可能需要多个Servlet合作，而这几个Servlet之间可以通过某种方式传递信息，但这个信息在请求结束后就无效了。request里的变量可以跨越forward前后的两页。但是只要刷新页面，它们就重新计算了。如果把变量放到request里，就说明它的作用域是request，它的有效范围是当前请求周期。 所谓请求周期，就是指从http请求发起，到服务器处理结束，返回响应的整个过程。在这个过程中可能使用forward的方式跳转了多个jsp页面，在这些页面里你都可以使用这个变量。

Servlet之间的信息共享是通过HttpServletRequest接口的两个方法来实现的：

```
void setAttribute (String name, Object value) //将对象value以name为名称保存到request作用域中。
Object getAttribute (String name)  //从request作用域中取得指定名字的信息。
```

JSP中的doGet()、doPost()方法的第一个参数就是HttpServletRequest对象，使用这个对象的setAttribute()方法即可传递信息。那么在设置好信息之后，要通过何种方式将信息传给其他的Servlet呢？这就要用到RequestDispatcher接口的forward()方法，通过它将请求转发给其他Servlet。

```
RequestDispatcher ServletContext.getRequestDispatcher(String path) //取得Dispatcher以便转发，path为转发的目的Servlet。
void RequestDispatcher.forward(ServletRequest request, ServletResponse response)//将request和response转发
```

因此，只需要在当前Servlet中先通过setAttribute()方法设置相应的属性，然后使用forward()方法进行跳转，最后在跳转到的Servlet中通过使用getAttribute()方法即可实现信息传递。

4.pageContext作用域

page对象的作用范围仅限于用户请求的当前页面，对于page对象的引用将在响应返回给客户端之后被释放，或者在请求被转发到其他地方后被释放。page里的变量只要页面跳转了，它们就不见了。如果把变量放到pageContext里，就说明它的作用域是page，它的有效范围只在当前jsp页面里。从把变量放到pageContext开始，到jsp页面结束，你都可以使用这个变量。

3. 代码示例：

两个页面03-四大作用域.jsp ,04-四大作用域02.jsp

03-四大作用域.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
 <title>Title</title>
</head>
<body>
 <%
     pageContext.setAttribute("name","张三");
     request.setAttribute("password","123456");
     session.setAttribute("sex","男");
     application.setAttribute("hobby","旅游，学习");
     //转发
request.getRequestDispatcher("04-四大作用域02.jsp").forward(request,response);
  %>
 <!--获取数据并输出-->
 <%
     Object name = pageContext.getAttribute("name");
     Object password = pageContext.getAttribute("password");
     Object sex = pageContext.getAttribute("sex");
     Object hobby = pageContext.getAttribute("hobby");
 %>
 当前页的pageContext中name值:<%=name%> <br>
 当前页request中password的值:<%=password%> <br>
 当前页session中sex的值:<%=sex%> <br>
 当前页application中hobby的值:<%=hobby%>
</body>
</html>
```

04-四大作用域02.jsp:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
 <title>Title</title>
</head>
<body>
 <!--获取保存在作用域对象中的数据-->
<%
 Object name = pageContext.getAttribute("name");
 Object password = request.getAttribute("password");
 Object sex = session.getAttribute("sex");
 Object hobby = application.getAttribute("hobby");
%>
 当前页的pageContext中name值:<%=name%> <br>
 当前页request中password的值:<%=password%> <br>
 当前页session中sex的值:<%=sex%> <br>
```

```
 当前页application中hobby的值:<%=hobby%>
</body>
</html>
```

作用域大小排序:
从小到大: pageContext<request<session<application

3.4 EL表达式

3.4.1 简介

EL（Expression Language） 是为了使JSP写起来更加简单。表达式语言的灵感来自于 ECMAScript 和 XPath 表达式语言，它提供了在 JSP 中简化表达式的方法，让Jsp的代码更加简化。

3.4.2 作用

是一种在JSP页面获取数据的简单方式(只能获取数据，不能设置数据)通过以下方式:
在JSP页面的任何静态部分均可通过：${expression}来获取到指定表达式的值
包含以下几种方式:
${绑定名} 获取javaBean对象或者其他类型数据
${绑定名.属性名} 获取javaBean中对应的属性名
${绑定名[索引值]} 获取集合或者数组中对应的值
${绑定名["key"]} 通过key的值获取在Map中对应的value值

同样也可以在el表达式内部做一些简单的运算:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
 <title>el基本运算</title>
</head>
<body>
  <!--基本运算-->
  加法运算:${3+2} <br>
  减法运算:${3-2} <br>
  乘法运算:${3*2} <br>
  除法运算:${3/2} <br>
  取余运算:${3%2} <br>
  <!--逻辑运算-->
  与:${5>4 && 7>8} <br>
  或:${5>4 || 7>8} <br>
  非:${! 5>4 } <br>
  非:${not (5>4) } <br>
  <!--三元运算-->
```

```
    三元运算:${5>4?"呵呵":"哈哈"}
  <!-- 空运算符 判断是否为空，如果为空则为true，否则为false-->
  ${empty uname}
</body>
</html>
```

3.4.4 原理

依次从四大作用域对象检索对应的绑定名，如果在某一个对象中获取到了，则不再去其他对象检索，否则继续检索，如果都没有查找到则返回的是一个空字符串，而不是null

如果如下操作时：以表达式${user.name}为例

EL表达式会根据name去User类里寻找这个name的get方法，此时会自动把name首字母大写并加上get前缀，一旦找到与之匹配的方法，El表达式就会认为这就是要访问的属性，并返回属性的值。

所以，想要通过EL表达式获取对象属性的值，那么这个属性就必须有与之对应的get方法。

可以通过指定检索四大作用域对象来提升检索速度，如下：

application-->applicationScope

session-->sessionScope

request-->requestScope

pageContext-->pageScope

比如：

${requestScope.user} 表示直接去request对象中获取user绑定名对应的值

3.4.4 案例

分别获取javaBean,javaBean属性,Map值，List，数组类型的值

代码示例：

ElDemoServlet

```
package com.ujiuye.servlet;

import com.ujiuye.bean.User;

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;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```java
@WebServlet("/elDemo")
public class ElDemoServlet extends HttpServlet {
 @Override
 protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
     //解决乱码问题的两行代码
     req.setCharacterEncoding("utf-8");
     resp.setContentType("text/html;charset=utf-8");
     //开始绑定数据
     //1.绑定一个字符串数据
     req.setAttribute("uname","邓紫棋");
     //2.绑定一个Map对象数据
     Map<String,String> map = new HashMap<>();
     map.put("name","张飒");
     req.setAttribute("map",map);
     //3.绑定一个List集合数据
     List<String> list = new ArrayList<>();
     list.add("三国演义");
     list.add("红楼梦");
     req.setAttribute("list",list);
     //4.绑定一个数组对象
     int[] arr ={12,89,10};
     req.setAttribute("arr",arr);
     //5.绑定一个javaBean对象
     User u = new User("李思思","666666");
     req.setAttribute("user",u);
     //转发
     req.getRequestDispatcher("06-el表达式常用方式.jsp").forward(req,resp);
 }
}
```

页面06-el表达式常用方式.jsp

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
 <title>el表达式常用方式</title>
</head>
<body>
 el获取字符串数据:${uname} <br>
 el获取Map对象中name对应的value值:${map["name"]} <br>
 el获取List集合中第二个数组:${list[1]} <br>
 el获取数组的第一个数据:${arr[0]} <br>
 el获取javaBeand的username属性值:${user.username}

</body>
</html>
```

3.5 JSTL标签库

3.5.1 简介

> JSTL（JavaServer Pages Standard Tag Library，JSP标准标签库）是一个不断完善的开放源代码的JSP标签库。它主要提供给JavaWeb开发人员一个标准通用的标签，开发人员可以利用这些标签取代JSP页面上的Java代码，从而提高程序的可读性，降低程序的维护难度。
>
> 需要注意的是：jstl中需要获取值的地方一般都要配合EL表达式去使用

3.5.2 使用

> 1. 下载导包：jstl.jar standard.jar 两个包
> 2. 将标签库引入 页面中
>
> ```
> <%-- 引入标签库 --%>
> <%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
> ```
>
> 3. 使用

```
1.<%--声明变量标签--%>
<c:set var="name" value="柳岩"></c:set>
 var:声明的变量名
 value:变量的值

2.<%--输出标签--%>
<c:out value="${name}" ></c:out>
value:要输出的变量的值,需要通过el表达式获取

3.<%--删除声明变量--%>
<c:remove var="name"></c:remove>
var:删除要删除的变量的名字

4.<%--if单分支标签--%>
<c:if test="${empty name}">测试if分支标签1</c:if>
test:当值为true的时候显示标签内部的值，否则不显示

5.<%--choose多分支标签--%>
<c:choose>
<c:when test="${1>2}">第一个条件的标签内容</c:when>
<c:when test="${3>2}">第二个条件的标签内容</c:when>
<c:otherwise>以上条件都不满足的标签内容</c:otherwise>
</c:choose>
当when标签的test的值为true的时候，显示标签内容，否则不显示，当多个when标签都为true的时候，值显示第一个为true的when标签内容，如果以上when标签都为false则显示otherwise的标签内容
注意：
when标签出现至少一个
otherwise标签最多出现一个
6.<%--集合或者数组的遍历标签--%>
```

```
<c:forEach items="" var="" >

</c:forEach>
```

items:要遍历的集合或者数组对象，需要通过el表达式获取

var：循环变量,可以理解为集合或者数组中的每一个值或者对象，名称自定义

代码示例：

JstlDemoServlet代码：

```
package com.ujiuye.servlet;

import com.ujiuye.bean.User;
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;
import java.util.ArrayList;
import java.util.List;

@WebServlet("/jstlDemo")
public class JstlDemoServlet extends HttpServlet {
 @Override
 protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
     //解决乱码问题的两行代码
     req.setCharacterEncoding("utf-8");
     resp.setContentType("text/html;charset=utf-8");
     User u1 = new User("赵丽颖","123456");
     User u2 = new User("倪妮","6688");
     List<User> list = new ArrayList<>();
     list.add(u1);
     list.add(u2);
     //将集合对象绑定到request对象上
     req.setAttribute("list",list);
     //转发
     req.getRequestDispatcher("07-jstl标签库.jsp").forward(req,resp);
 }
}
```

07-jstl标签库.jsp页面代码：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
```

```
 <title>jstl标签库</title>
</head>
<body>
 <%--声明变量标签--%>
 <c:set var="name" value="柳岩"></c:set> <br>
 <%--输出标签--%>
 <c:out value="${name}" ></c:out> <br>
 <%--删除声明变量--%>
 <c:remove var="name"></c:remove> <br>
 当删除name变量后通过el表达式获取的值:${name} <br>
 <%--if 分支标签--%>
 <c:if test="${empty name}">测试if分支标签1</c:if> <br>
 <c:if test="${!empty name}">测试if分支标签2</c:if> <br>
 <%--choose多分支标签--%>
 <c:choose>
     <c:when test="${1>2}">第一个条件的标签内容</c:when>
     <c:when test="${3>2}">第二个条件的标签内容</c:when>
     <c:otherwise>以上条件都不满足的标签内容</c:otherwise>
 </c:choose>
 <%--集合或者数组的遍历标签--%>
 <c:forEach items="${list}" var="u" >
     用户名:${u.username} <br>
     密  码:${u.password} <br>
 </c:forEach>
</body>
</html>
```