# 15.JSP

## 15.1.jsp概述

**jsp介绍**

由SUN公司提供的动态web资源的开发技术, 看起来非常像html, 但是可以在JSP页面中写java代码, 所以JSP是一种动态web资源开发技术.

**JSP技术的由来**

JSP的出现是为了解决Servlet在响应时不适合向外输出页面的问题。

Servlet本质上是一段java代码, 非常适合处理逻辑, 但是不合适向外输出页面。

HTML适合用来开发页面, 展示数据, 但是HTML开发出来的页面本质上就是一个文档, 无法展示动态的数据。

JSP非常适合编写HTML代码, 适合作为响应页面向外输出, 同时JSP里可以写java代码, 也可以展示动态的数据。

所以JSP可以解决既要处理逻辑又要向外输出页面的难题。

JSP在第一次访问时, 会被翻译成一个Servlet, 对JSP访问后看到的页面 其实就是翻译后的Servlet在向外输出！！

## 15.2.JSP语法

**模版元素**

直接写在jsp页面中的html内容称之为jsp页面中的模版元素

模版元素在翻译过来的Servlet中被out.write()原样输出到浏览器中

out.write("\tday13....index.jsp........\r\n");

例如：

|  |
| --- |
| 创建web项目day13，将WebRoot下的index.jsp中第一行编码改为utf-8，然后部署并启动服务器后访问该页面，可以在tomcat目录\work\Catalina下对应虚拟主机中找到被翻译的servlet。 |

**JSP表达式**

<%= 脚本表达式 %> 常量 变量 表达式(3+4)

在翻译过来的Servlet中, 计算表达式的值原样输出

out.print( 100+150 );

例如：

|  |
| --- |
| 在index.jsp中<body>标签内添加如下代码：  <h1>JSP表达式</h1>  <%= "abc" %>  <%= 100+150 %>  <%= request.getContextPath() %>  <% String username = "张飞"; %>  <%= username %> |

**JSP脚本片段**

<% 若干java语句 %>

在翻译过来的servlet中, 脚本片段被复制粘贴到对应位置执行

翻译前:

<% for(int i=0; i<5; i++){ %>

Hello JSP~~~~~<br/>

<% } %>

翻译后:

for(int i=0; i<5; i++){

out.write("\r\n");

out.write("\t\tHello JSP~~~~~<br/>\r\n");

out.write("\t");

}

多个脚本片段之间的变量可以互相访问

在某一个脚本片段中的java代码可以是不完整的, 但是要求在翻译过来的servlet中整体的代码必须是完整符合java语法的

任何文本，HTML标记，JSP元素必须在脚本片段之外

例如：

|  |
| --- |
| 在index.jsp中<body>标签内添加如下代码：  <h1>JSP脚本片段</h1>  <% **for**(**int** i=0; i<5; i++){ %>  Hello JSP~~~~~<br/>  <% } %>  <% **int** i = 10; %>  <% i= i+5; %>  <%= i %> |

**JSP声明:(不常用)**

格式: <%! java代码 %>

写在jsp声明中的内容, 在翻译过来的servlet中会和servlet方法平级成为类的成员

写在JSP脚本片段中的内容, 在翻译过来的servlet中是方法的局部变量!

**JSP注释**

格式: <%-- JSP注释 --%>

<%-- out.write("aaa"); --%> 被JSP注释注释的内容, 在翻译的过程中被抛弃, 不会被翻译

<% //out.write("bbb"); %> 被java注释注释的内容, 在翻译的过程中被当作脚本片段翻译到servlet中, 但是由于被注释了, 所以也不会执行

<!-- <% out.write("ccc"); %> --> 被HTML注释注释的内容, 在翻译过程中直接当作模版元素原样输出到浏览器, 但是浏览器认为是注释, 所以不予显示

!!!在jsp中最好使用jsp注释, html注释慎用!!

例如：

|  |
| --- |
| 在index.jsp中<body>标签内添加如下代码：  <h1>JSP注释</h1>  <%-- <% out.write("aaa"); %> --%>  <% //out.write("bbb"); %>  <!-- <% out.write("ccc"); %> -->    <% String name = "小兰兰"; %>  <!-- <% name = "凤姐"; %> -->  <%= name %> |

**JSP指令**

<%@ 指令名称 若干属性声明... %>

-- 不会直接产生输出, 用来指挥JSP解析引擎如何来翻译当前JSP页面中其他部分的内容

**page指令**

-- 用来声明当前JSP页面的基本属性的, page指令可以写在JSP页面的任意位置, 但是为了可读性考虑, 一般情况下最好放在JSP页面的最前面

格式: <%@ page ... %>

page指令属性介绍

~~(1) [ language="java" ]

当前JSP使用的开发语言

~~(2) [ extends="package.class" ]

当前JSP翻译成servlet后要继承的类，注意此值必须是一个servlet的子类，一般情况下不要改

!!(3) [ import="{ package.class | package.\*}, ..." ]

导入需要使用到的包

java.lang.\*;

javax.servlet.\*;

javax.servlet.JSP.\*;

javax.servlet.http.\*;

可以在一个page指令的import属性中导入多个包 也可以通过多个page指令来分别导包

示例:

<%@ page language="java" import="java.util.\*,java.sql.\*" pageEncoding="utf-8" isErrorPage="true"%>

<%@ page import="java.io.\*" %>

!(4) [ session="true | false" ]

用来指定当前页面是否使用session，如果设置为true，则翻译过来的servlet中将会有对session对象的引用，于是可以直接在JSP中使用session隐式对象。但是这将导致一旦访问JSP就会调用request.getSession()方法，可能导致不必要的空间浪费。如果确定JSP中不需要session可以设为false

(5) [ errorPage="relative\_url" ]

如果页面出错，将要跳转到的页面（如：errorPage="/err/error.jsp"），除了在JSP中使用此属性指定错误页面外也可以在web.xml中配置整个web应用的错误页面，如果两个都设置则JSP中的此属性起作用

配置示例：

<error-page>

<error-code>500</error-code>

<location>/err/500.jsp</location>

</error-page>

<error-page>

<error-code>404</error-code>

<location>/err/404.jsp</location>

</error-page>

(6) [ isErrorPage="true | false" ]

如果设置此属性为true , 翻译过来的servlet中将会含有Exception隐式对象,其中封装的就是上一个页面中抛出的异常信息

!!(7) [ pageEncoding="characterSet | ISO-8859-1" ]

pageEncoding有4个功能：

1. jsp保存数据时使用的编码（MyEclipse会识别此属性并设置编码）
2. Jsp在翻译成servlet时使用的编码
3. Servlet（服务器）在向浏览器发送数据时使用的编码
4. 通知浏览器使用哪一个编码来接受服务器发送过来的数据

此属性一旦设置好,翻译引擎会间接帮我们设置content-type属性。但是请求参数乱码依然需要设置，因为请求参数并非传输到jsp翻译的servlet中。

1. [ buffer="none | 8kb | sizekb" ] out隐式对象所使用的缓冲区的大小

(9) [ autoFlush="true | false" ] out隐式对象是否自动刷新缓冲区，默认为true，不需要更改

**include指令**

<%@ include file="" %> -- 可以实现页面包含的效果

（1）include指令实现的包含叫做静态包含:

多个JSP文件翻译成一个servlet, 最终由这一个servlet向外输出数据, 这是源文件级别的包含

（2）其他方式的包含叫做动态包含:

被包含的页面各自翻译成servlet, 包含的过程其实就是各个servlet分别执行后在输出流上的合并

总结: 静态包含在效率上更高一些, 尽量使用静态包含.

例如：

|  |
| --- |
| 在WebRoot下创建文件夹include，并创建\_head.jsp，设置为如下内容：  <%@ page language=*"java"* errorPage=*"/err/error.jsp"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>  <!DOCTYPE HTML>  <html>  <head>  <title>My JSP 'index.jsp' starting page</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  </head>  <body>  \_head.jsp...<br/> |
| 在include下，创建\_foot.jsp，设置为如下内容：  <%@ page language=*"java"* errorPage=*"/err/error.jsp"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>  \_foot.jsp...<br/>  </body>  </html> |
| 在include下，创建body1.jsp，设置为如下内容：  <%@ page language=*"java"* errorPage=*"/err/error.jsp"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>  <%-- 包含头部 --%>  <%@ include file=*"/include/\_head.jsp"* %>  body1...<br/>  <%-- 包含尾部 --%>  <%@ include file=*"/include/\_foot.jsp"* %> |
| 在include下，创建body2.jsp，设置为如下内容（注意，需要在page标签中设置属性buffer="0kb"）：  <%@ page language=*"java"* errorPage=*"/err/error.jsp"* buffer=*"0kb"* import=*"java.util.\*"* pageEncoding=*"utf-8"*%>  <% request.getRequestDispatcher("/include/\_head.jsp").include(request, response); %>  body2...<br/>  <% request.getRequestDispatcher("/include/\_foot.jsp").include(request, response); %> |

**taglib指令**

导入tld标签库文件

JSTL标签库

## 15.3.JSP的九大隐式对象

JSP翻译引擎在将JSP翻译成servlet的过程中, 在servlet里预先定义了九个对象, 因此我们可以在JSP页面中直接使用这九个对象

page(this) 与Servlet对应

!!request 与request对应

!!response 与response对应

config 与ServletConfig对应

application 与ServletContext对应

!!session 与Session对应

exception 代表异常的对象

!out

!pageContext

其中其他的七个我们都学习过了，只需要再学习一下out和pageContext即可。

### 15.3.1.out隐式对象

相当于response.getWriter();

区别在于out对象自带缓冲区, 如果out和response.getWriter()混用可能会造成输出顺序上混乱（可参见图片“out和response混用顺序问题.jpg”）

page指令中buffer就是用来设置out缓冲区的。

总结起来一句话, 如果想要在JSP页面中输出内容, 直接使用out, 尽量别使用response.getWriter();

### 15.3.2.pageContext隐式对象

代表当前JSP页面的运行环境的对象

PageContext对象功能:

(1) 作为入口对象获取其他八大隐式对象

getPage()方法 返回page隐式对象

getRequest()方法 返回request隐式对象

getResponse()方法 返回response隐式对象

getServletConfig()方法 返回config隐式对象

getServletContext()方法 返回application隐式对象

getSession()方法 返回session隐式对象

getException()方法 返回exception隐式对象

getOut()方法 返回out隐式对象

其他八大对象也可以直接使用，但是在以后el表达式中只能通过pageContext获取。

(2) 本身也是一个域对象, 也可以作为入口对象来操作其他三大作用域中的数据

a)本身是个域对象，并提供了方法操作域中的属性

setAttribute(String name, Object obj);

getAttribute(String name);

removeAttribute(String name);

getAttributeNames();

生命周期: 访问JSP页面开始时创建, 访问JSP页面结束时销毁

作用范围: 当前JSP页面

主要功能: 在当前JSP页面中共享数据

四大域对象的作用范围大小排序：ServletContext > Session > request > pageContext

b)作为入口对象, 可以操作其他三大作用域

setAttribute(String name, Object value,int scope)

getAttribute(String name,int scope)

removeAttribute(String name,int scope)

其中pageContext中代表域的常量：

PageContext.APPLICATION\_SCOPE

PageContext.SESSION\_SCOPE

PageContext.REQUEST\_SCOPE

PageContext.PAGE\_SCOPE

例如：

pageContext.setAttribute("addr", "北京");

pageContext.setAttribute("addr", "上海",pageContext.REQUEST\_SCOPE);

pageContext.setAttribute("addr", "广州",pageContext.SESSION\_SCOPE);

pageContext.setAttribute("addr", "深圳",pageContext.APPLICATION\_SCOPE);

pageContext对象还额外提供了findAttribute(String name)方法，该方法会按照由小到大的顺序在四大作用域中搜寻指定名称的属性, 如果找到就返回, 如果都找不到就返回一个null

例如：pageContext.findAttribute("addr")

(3) 提供了便捷方法实现请求转发和包含

下面是通过request对象实现请求转发或包含：

request.getRequestDispatcher("/7.jsp").forward(request, response);

request.getRequestDispatcher("/7.jsp").include(request, response);

或通过page指令实现包含：

<%@include file="xxx.jsp" %>

pageContext对象实现请求转发或包含（转发时，转发页面不能设置buffer="none"）：

转发: pageContext.forward("/index.jsp");

包含: pageContext.include("/index.jsp");

## 15.4.四大作用域的比较

**ServletContext:** 代表整个WEB应用的对象, 在整个WEB应用范围内都可以访问到.

生命周期:

在服务器加载WEB应用之后立即创建, WEB应用被移出容器时销毁

作用范围:

整个WEB应用

主要功能: 在整个WEB应用范围内共享数据

**Session:**

生命周期:

第一次调用request.getSession()方法时创建, 销毁包括超时(如果30分钟不使用session)、自杀（调用invalidate方法）、意外身亡（当服务器意外关闭时）！

作用范围:

一次会话范围内

主要功能:

在整个会话范围内实现数据的共享

**request:** 代表HTTP请求的对象

生命周期:

一次请求开始时创建, 一次请求结束后销毁

作用范围:

整个请求链

主要功能:

在一次请求中共享数据(注册失败后转发回注册页面提示错误消息)

**pageContext:** 代表JSP运行环境的对象

生命周期:

在开始访问JSP时创建, 访问JSP结束时销毁!

作用范围:

在整个JSP范围内

主要功能:

在整个JSP范围内共享数据

## 15.5.JSP标签技术简介

在JSP页面中写入大量的java代码会导致JSP页面中html代码和java代码混杂在一起, 会造成页面非常的混乱, 难于维护

于是在JSP的2.0版本中, sun提出了JSP标签技术, 推荐使用标签来代替JSP页面中java代码, 并且推荐, JSP2.0以后不要在JSP页面中出现任何一行java代码。

**JSP标签**

sun公司开发的JSP页面的标签技术, 包含了不少标签, 但是不太好用, 我们只要求掌握三个

<jsp:forward> 用来替代request.getRequestDispatcher().forward()

<jsp:include> 用来替代request.getRequestDispatcher().include()

<jsp:param> 配合前两个标签使用，可以在包含或转发时，带一些参数过去

例如：

|  |
| --- |
| 在day13中WebRoot下创建文件夹：jsptag，并创建01.jsp，<body>中添加如下内容：  <body>  This is 01.jsp... <br>  <!-- 请求转发 -->  <%-- <jsp:forward page="/jsptag/02.jsp"></jsp:forward> --%>  <!-- 请求包含 -->  <jsp:include page=*"/jsptag/02.jsp"*>  <!-- 包含时带参数过去 -->  <jsp:param value=*"abc"* name=*"username"*/>  </jsp:include>  </body> |
| Jsptag中创建02.jsp，<body>中添加如下内容：  <body>  This is 02.jsp... <br>  <%= request.getParameter("username") %>  </body> |

## 15.6.EL表达式

EL表达式, 可以非常方便替代JSP页面中的JSP表达式(<%= %>)

基本结构: ${ 表达式 } 例如：${ param.username }

EL只能获取不能设置!!!

EL只能获取不能遍历!!!

EL表达式提供了如下功能：

**（1）获取数据**

获取常量: 支持 数字/字符串/布尔值, 整个el表达式的值就是当前常量的值

例如：${ "abc" } ${ 100+23 }

获取变量: el表达式可以获取变量的值, 在获取时它的底层会调用pageContext的findAttribute()方法按照由小到大的顺序在四大作用中搜寻指定名称的值, 如果找到就直接返回该值, 如果找不到就什么也不输出

例如：<% pageContext.setAttribute("str", "hello el..."); %> ${ str }

获取数组中的数据: 可以通过 数组变量名[数字] 来获取域中数组中的内容

例如：

<% String[] strs = {"张三","李四","王五"};

pageContext.setAttribute("strs",strs); %>

${ strs[0] }

获取集合中的数据: 可以通过 集合变量名[数字] 来获取域中集合中的内容

例如：${ list[3] }

获取map中的数据: 可以通过 map变量名[键] 或者是 map变量名.键 获取map中的数据

例如：${ map.addr } 或 ${ map["addr"] }

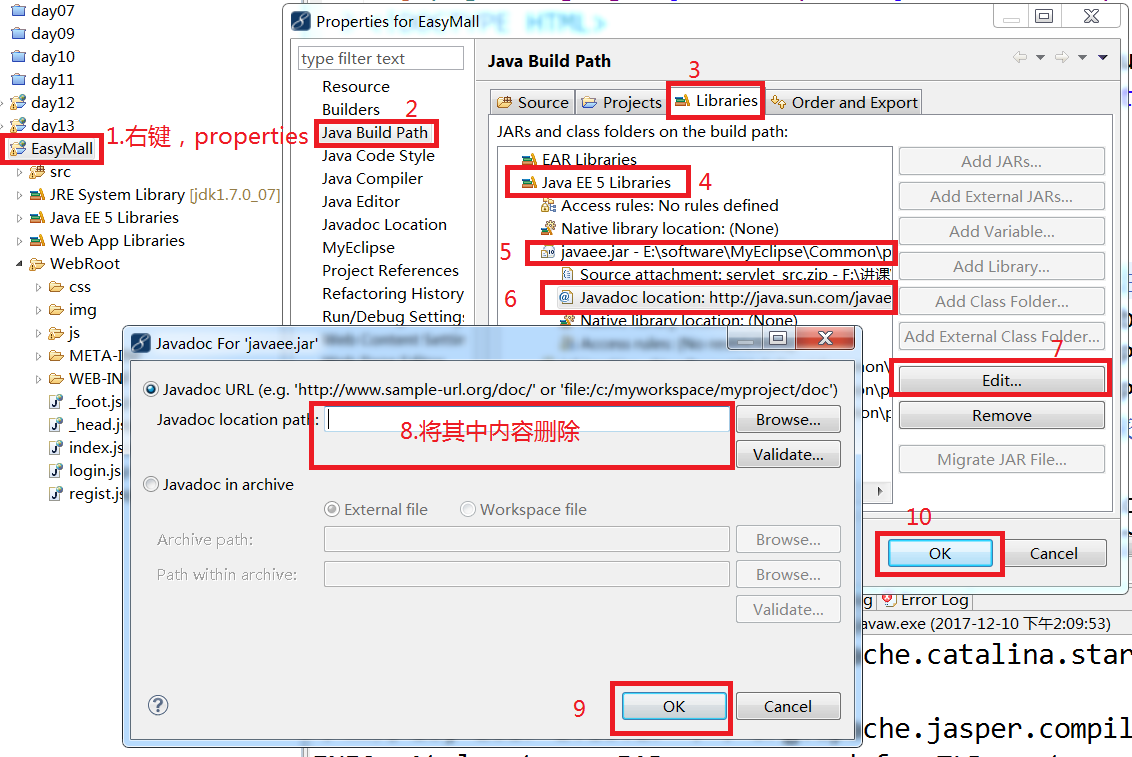
获取javaBean中的属性: 在获取javaBean中的属性值时, 点什么就相当于get什么(比如: p.username <==> p.getUsername())

## 之前获取web应用的名称是通过 <%= request.getContextPath() %>

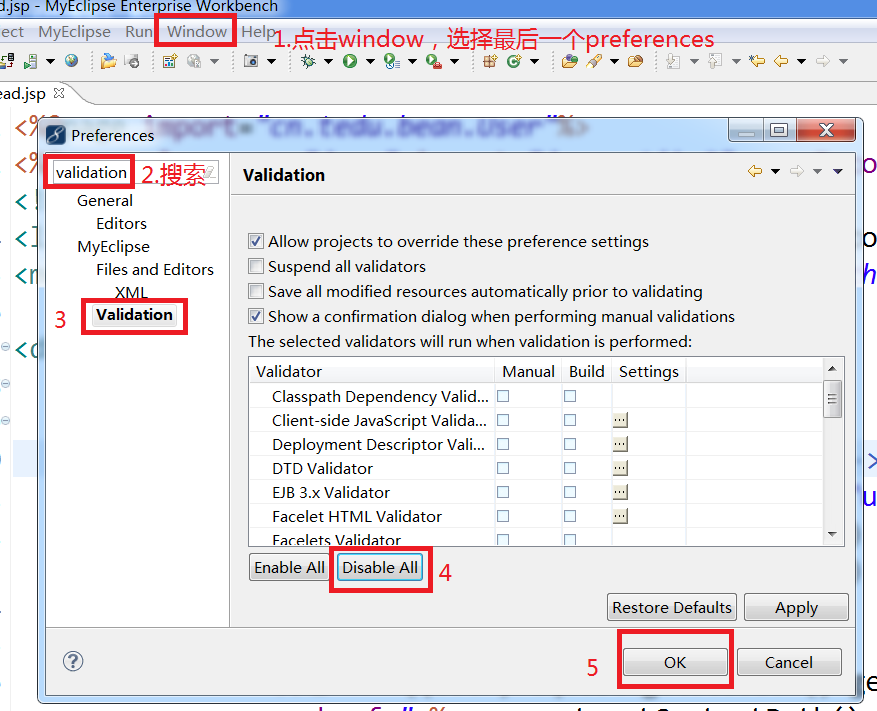
但是这种写法不推荐, 推荐使用el表达式来获取: ${ pageContext.request.contextPath }

当写此表达式的时候，可能会出现卡顿现象，可以使用如下步骤解决：

第一步：



第二步：



1. **执行运算**

**算术运算**

在El表达式中, 加号只是一个运算符, 不能作为连接符使用.

在进行算术运算时, 对于非数字会试图先转换成数字再参与运算, 如果转换不成功就会抛异常!!

**关系运算**

关系运算符得到的是boolean值

> gt

< lt

>= ge

<= le

!= ne

== eq

**逻辑运算**

逻辑运算符得到的是boolean值

&& and

|| or

! Not

**三元表达式**

表达式 ? "xx" : "xx"

**empty运算:**

判断对象是否为null,

判断字符串是否为空字符串,

判断数组是否为空数组,

判断集合中是否没有任何元素,

判断域对象中是否没有任何属性

例如：${ empty str } ${ empty strs } ${ empty list } ${ empty map }

${ empty pageScope} ${ empty requestScope }

${ empty sessionScope } ${ empty applicationScope }

**（3）获取常用开发对象**

el中内置了11个常用对象, 可以不用预先存入域中就可以使用

代表当前JSP页面环境的pageContext对象

（1）!!!pageContext - 有了它就意味着el中就有了其他八大隐式对象

代表四大作用域的隐式对象, 用来明确指定从哪个域中查找数据

（2）!!!pageScope -- pageContext

（3）!!!requestScope -- request

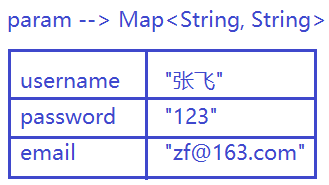
（4）!!!sessionScope -- session

（5）!!!applicationScope -- ServletContext

注意: 上面这四个对象只能代表四大作用域对象身上的那个域!!

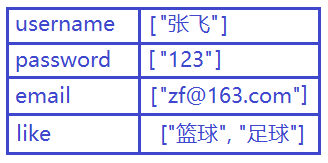
请求参数组成的map

（6）!!!param - Map<String, String>



例如: 获取 ”张飞” : ${ param.username } ${ param[“username”] }

（7）paramValues - Map<String, String[]>



例如: 获取 ”张飞” : ${ paramValues.username[0] }

${ paramValues[“username”][0] }

请求头组成的map

（8）header -- Map<String, String>

（9）headerValues - Map<String, String[]>

key为请求头的名字

host: localhost

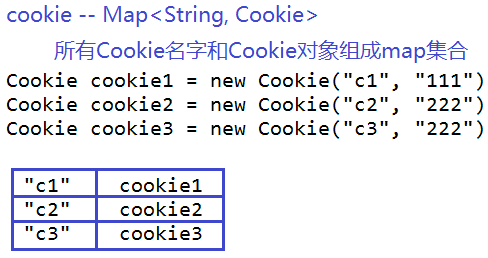
header.host

headerValues.host[0]

所有cookie信息组成的map

（10）!!!cookie -- Map<String, Cookie>

key为Cookie的名字, 值为Cookie对象

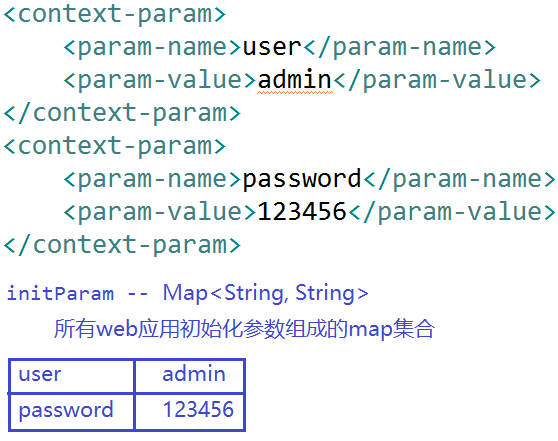


例如: 获取”111” ${ cookie.c1.value }

所有web应用初始化信息组成的map集合

（11）initParam -- Map<String, String>

key初始化参数的名字, value为参数对应的值



例如: 获取用户名和密码: ${ initParam.user } ${ initParam.password }

## 15.7.JSTL标签库

JSTL：通用的JSP标签库 提供了实现java代码中最常见功能的操作标签(taglib指令)

为javaweb开发人员提供的一套标准通用的标签库, 可以和el配合起来替JSP页面中的大部分的java代码

javaee4.0及其之前的版本中, 在使用前需要导入jstl开发包, 在javaee5.0及其以后版本中内置了jstl开发包, 所以不需要我们手动导入就可以直接使用了!!

JSTL标签库子库：

**核心标签库 (core) --- c**

国际化标签 fmt

数据库标签 sql --Servlet JSP

XML标签 xml

JSTL函数(EL函数) fn

**JSTL核心标签库：**

首先，需要导入标签库：

在jsp最上面加上如下代码：

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

~~<c:out> 标签用于输出一段文本内容到pageContext对象当前保存的 “out”对象中（很少用）。

输出常量、变量、表达式，例如：

<c:out value=”aaa”></c:out>

<%

String str = "hello jstl…";

pageContext.setAttribute("str", str);

%>

<c:out value="${ str }"></c:out>

<c:out value="${3+5}"></c:out>

**!!!<c:set>** 标签用于把某一个对象存在指定的域范围内，或者设置Web域中的java.util.Map类型的属性对象或JavaBean类型的属性对象的属性。

**往四大域中存入属性**

<c:set var="username" value="张无忌" scope="request"></c:set>

${ username }

其中var用来声明属性的名字, value用来指定属性的值

scope指定将属性存入哪一个域中，不加的话，默认是在pageScope中。当指定时，值无需写scope，例如四个域分别是：

page/request/session/application

**修改四大域中的属性**

修改域中map的属性:

<c:set target="${ map }" property="username" value="张三丰"></c:set>

修改域中javaBean的属性:

<c:set target="${ person }" property="age" value="38"></c:set>

其中 target属性用于指定修改域中的哪一个属性, 比如map或JavaBean

property属性用于指定修改map或JavaBean中的哪一个属性

value属性用于指定修改的值是什么

<c:remove> 标签用于删除各种Web域中的属性

例如: <c:remove var="person" scope="request"/>

scope为可选属性，可以指定在哪一个域中删除。

<c:catch> 标签用于捕获嵌套在标签体中的内容抛出的异常，其语法格式如下：<c:catch [var="varName"]>nested actions</c:catch>

例如：<c:catch var="e"><% int i = 1/0; %></c:catch> ${ e }

其中，属性var中为保存异常信息的对象

**!!!<c:if test="">** 标签可以构造简单的“if-then”结构的条件表达式

例如：<c:if test="${ empty sessionScope.user }">您还没有登录</c:if>

<c:if test="${ !(empty sessionScope.user) }">您已经登录</c:if>

**!!!<c:choose>** 标签用于指定多个条件选择的组合边界，它必须与<c:when>和<c:otherwise>标签一起使用。使用<c:choose>，<c:when>和<c:otherwise>三个标签，可以构造类似 “if-else if-else” 的复杂条件判断结构。

例如：<c:set var="day" value="4"></c:set>

<c:choose>

<c:when test="${ day == 1 }">星期一</c:when>

<c:when test="${ day == 2 }">星期二</c:when>

<c:when test="${ day == 3 }">星期三</c:when>

<c:when test="${ day == 4 }">星期四</c:when>

<c:when test="${ day == 5 }">星期五</c:when>

<c:otherwise>周末</c:otherwise>

</c:choose>

**!!!<c:forEach>** 标签用于对一个集合对象中的元素进行循环迭代操作，或者按指定的次数重复迭代执行标签体中的内容。

例如：遍历20内的奇数，若数字所在的位置是3的倍数，显示红色

<c:forEach begin="1" end="20" step="2" var="i" varStatus="status">

<c:if test="${ status.count % 3 == 0 }">

<span style="color:red">${ i }</span>

</c:if>

<c:if test="${ status.count % 3 != 0 }">

${ i }

</c:if>

${ status.last ? "" : "," }

</c:forEach>

//1 , 3 , 5 , 7 , 9 , 11 , 13 , 15 , 17 , 19

begin：指定开始的值

end：指定结束的值

step：指定步长

var：当前迭代的元素，会保存到pageScope域中

varStatus：保存迭代信息的对象，属性有：

first：当前遍历元素是否为第一个，是则返回true，反之为false。

last：当前遍历元素是否为最后一个，是则返回true，反之为false。

index：获取当前被遍历元素的索引，从0开始。

count：获取当前被遍历的元素是第几个，从1开始。

遍历数组

<c:forEach items="${ strs }" var="str">

${ str }

</c:forEach>

遍历map

<c:forEach items="${ map }" var="entry">

${ entry.key } : ${ entry.value } <br/>

</c:forEach>

items：将要遍历的集合对象

<c:forTokens> 用来浏览一字符串中所有的成员，其成员是由定义符号所分隔的

例如：

<% String str1 = "www.tedu.com.cn";

pageContext.setAttribute("str1", str1);%>

<c:forTokens items="${ str1 }" delims="." var="str">${ str } </c:forTokens>

//www tedu com cn

<c:import> 标签,实现include操作

例如：<c:import url="/index.jsp"></c:import>

<c:redirect>标签用于实现请求重定向

例如：<c:redirect context="${ pageContext.request.contextPath }" url="/index.jsp"></c:redirect>

<c:param>标签 在重定向或者请求包含时，可以携带一些参数。<c:param>标签可以嵌套在<c:import>或<c:redirect>标签内。

例如：

<c:redirect context="${ pageContext.request.contextPath }" url="/index.jsp">

<c:param name="username" value="admin"></c:param>

</c:redirect>

可以通过EL获取：${ param.username }

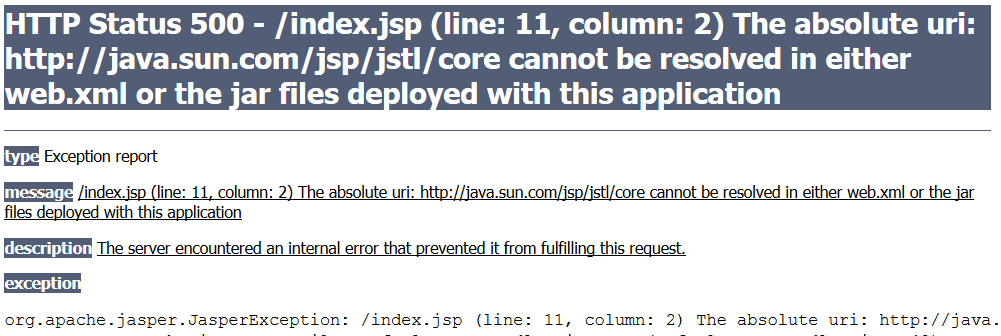
~~<c:url>进行url重写的标签，url重写由于成本太高，几乎不用了!!!

## 15.8.EL+JSTL改造EasyMall项目

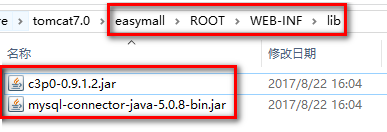
|  |
| --- |
| 1.改造\_head.jsp  首先，将所有<%= request.getContextPath() %>替换为：（共6处）  ${ pageContext.request.contextPath }  其次，引入标签库，在jsp最上面加入如下代码：（注意，还需要导包）  <%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"* %>  然后，修改登录/注册部分内容  <!-- 如果用户未登录，则提示用户登录或注册 -->  <c:if test="${ **empty** sessionScope.user }">  <a href=*"*${ pageContext.request.contextPath }*/login.jsp"*>登录</a>&nbsp;&nbsp;|&nbsp;&nbsp;  <a href=*"*${ pageContext.request.contextPath }*/regist.jsp"*>注册</a>  </c:if>  <!-- 如果用户已登录，则提示欢迎xxx回来 -->  <c:if test="${ !(**empty** sessionScope.user) }">  欢迎 ${ user.username } 回来，  <a href=*"*${ pageContext.request.contextPath }*/servlet/LogoutServlet"*>退出</a>  </c:if> |
| 1. 改造index.jsp   将所有<%= request.getContextPath() %>替换为：（共14处）  ${ pageContext.request.contextPath } |
| 3.改造\_foot.jsp  将所有<%= request.getContextPath() %>替换为：（共1处）  ${ pageContext.request.contextPath } |
| 4.改造regist.jsp  首先，将所有<%= request.getContextPath() %>替换为：（共6处）  ${ pageContext.request.contextPath }  然后，将<table>中第一个<td>中，获取msg的地方用el，改为：${ msg }  同样，在各<input>中，value中回显参数的部分也用el代替，例如：  <input type=*"text"* name=*"username"* value=*"*${ param.username }*"*/> |
| 5.改造login.jsp  首先，将所有<%= request.getContextPath() %>替换为：（共2处）  ${ pageContext.request.contextPath }  然后，将<table>中第一个<td>中，获取msg的地方用el，改为：${ msg }  然后，将获取cookie部分代码删除（或注释掉），并将用户名对应<input>中的value改为通过el获取：  <input type=*"text"* name=*"username"* value=*"*${ cookie.remname.value }*"*/>  将记住用户名对应的<input>中java代码部分改为：  ${ empty cookie.remname ? "" : "checked=*'checked'*" }  最后，发现记住的用户名没有解码，在<head>标签中加入如下代码：  <script type=*"text/javascript"*>  /\* 使用js代码对用户名进行url解码 \*/  window.onload = **function**(){  **var** oInp = document.getElementsByName("username")[0];  oInp.value = decodeURI(oInp.value);  }  </script> |

**问题：**

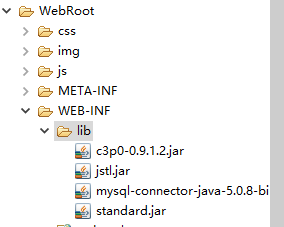
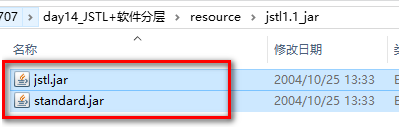
如果再jsp中使用了jstl标签库, 出现如下错误:



原因在于, 在将EasyMall项目发布到[www.easymall.com](http://www.easymall.com)主机时, 并没有将jstl相关的包发布过去, 所以才导致上面的错误



解决方案: 自己手动将jstl的包（可以在tomcat中其他项目的lib包下找到，如：[tomcat7.0]\webapps\day13\WEB-INF\lib\jstl-1.2.jar或者课前资料中也可以）拷贝到EasyMall项目中的lib目录下, 再次发布项目, 重新启动服务器即可!!



## 15.9.作业练习

1. JSP和Servlet是什么关系？

答：JSP在本质上就是Servlet，但是两者的创建方式不一样。

Servlet完全是JAVA程序代码构成，擅长于流程控制和事物处理，而通过Servlet来生成动态网页很不直观;

JSP由HTML代码和JSP标签构成，可以方便地编写动态网页，因此在实际应用中采用Servlet来控制业务流程，而采用JSP来生成动态网页。

2. JSP 请求是如何被处理的？

答：浏览器首先要请求一个以.jsp 扩展名结尾的页面，发起 JSP 请求，然后， Web 服务器读取这个请求，使用 JSP 编译器把 JSP 页面转化成一个 Servlet 类。需要注意的是，只有当第一次请求页面或者是 JSP 文件发生改变的时候 JSP 文件才会被编译，然后服务器调用 servlet 类，处理浏览器的请求。一旦请求执行结束， servlet 会把响应发送给客户端。

3. JSP有哪些内置对象？作用分别是什么？

答：request代表的是请求对象,可以用于获得客户机的信息,也可以作为域对象来使用，使用request保存的数据在一次请求范围内有效。

response是响应对象,代表的是从服务器向浏览器响应数据.

Session代表的是一次会话，可以用于保存用户的私有的信息,也可以作为域对象使用，使用session保存的数据在一次会话范围有效

Application：代表整个应用范围,使用这个对象保存的数据在整个web应用中都有效。

Page指的是JSP被翻译成Servlet的对象的引用.

Config：指的是ServletConfig，用于JSP翻译成Servlet后获得Servlet的配置的对象.

Exception:在页面中设置isErrorPage=”true”，即可使用，是Throwable的引用，用来获得页面的错误信息。

Out:是用于向页面输出内容的对象

pageContext对象可以用来获得其他8个内置对象,还可以作为JSP的域范围对象使用。pageContext中存的值是当前的页面的作用范围。

4. 讲解JSP中的四种作用域

答：pageContext：作用域仅限于当前页面对象，可以近似于理解为java的this对象，离开当前JSP页面（无论是redirect还是forward），则pageContext中的所有属性值就会丢失。

request：作用域是同一个请求之内，在页面跳转时，如果通过forward方式跳转，则forward目标页面仍然可以拿到request中的属性值。如果通过redirect方式进行页面跳转，由于redirect相当于重新发出的请求，此种场景下，request中的属性值会丢失。

session：session的作用域是在一个会话的生命周期内，会话失效，则session中的数据也随之丢失。

application：作用域是最大的，只要服务器不停止，则application对象就一直存在，并且为所有会话所共享

5. 在四大域中搜寻属性，搜寻的顺序是什么？

答：按照由小到大的顺序在四大作用域中搜寻指定名称的属性，即：pageContext-request-session-application

6. EL表达式中有哪些隐含对象可用？

答：el中内置了11个常用对象，分别是：

代表当前JSP页面环境的pageContext对象

（1）pageContext - 有了它就意味着el中就有了其他八大隐式对象

代表四大作用域的隐式对象, 用来明确指定从哪个域中查找数据

（2）pageScope

（3）requestScope

（4）sessionScope

（5）applicationScope

请求参数组成的集合

1. param
2. paramValues

请求头组成的集合

（8）header

（9）headerValues

所有cookie信息组成的集合

1. cookie

所有web应用初始化信息组成的集合

1. initParam