# day12 自定义标签JSTL标签库,java web之设计模式

## JSP指令

###### JSP指令概述

JSP指令的格式：<%@指令名 attr1=”” attr2=”” %>，一般都会把JSP指令放到JSP文件的最上方，但这不是必须的。

JSP中有三大指令：page、include、taglib，最为常用，也最为复杂的就是page指令了。

###### 2　page指令

page指令是最为常用的指定，也是属性最多的属性！

page指令没有必须属性，都是可选属性。例如<%@page %>，没有给出任何属性也是可以的！

在JSP页面中，任何指令都可以重复出现！

<%@ page language=”java”%>

<%@ page import=”java.util.\*”%>

<%@ page pageEncoding=”utf-8”%>

这也是可以的！

2.1　page指令的pageEncoding和contentType（重点）

pageEncoding指定当前JSP页面的编码！这个编码是给服务器看的，服务器需要知道当前JSP使用的编码，不然服务器无法正确把JSP编译成java文件。所以这个编码只需要与真实的页面编码一致即可！在MyEclipse中，在JSP文件上点击右键，选择属性就可以看到当前JSP页面的编码了。

contentType属性与response.setContentType()方法的作用相同！它会完成两项工作，一是设置响应字符流的编码，二是设置content-type响应头。例如：<%@ contentType=”text/html;charset=utf-8”%>，它会使“真身”中出现response.setContentType(“text/html;charset=utf-8”)。

无论是page指令的pageEncoding还是contentType，它们的默认值都是ISO-8859-1，我们知道ISO-8859-1是无法显示中文的，所以JSP页面中存在中文的话，一定要设置这两个属性。

其实pageEncoding和contentType这两个属性的关系很“暧昧”：

* 当设置了pageEncoding，而没设置contentType时： contentType的默认值为pageEncoding；
* 当设置了contentType，而没设置pageEncoding时： pageEncoding的默认值与contentType；

**也就是说，当pageEncoding和contentType只出现一个时，那么另一个的值与出现的值相同。如果两个都不出现，那么两个属性的值都是ISO-8859-1。所以通过我们至少设置它们两个其中一个！**

2.2　page指令的import属性

import是page指令中一个很特别的属性！

import属性值对应“真身”中的import语句。

import属性值可以使逗号：<%@page import=”java.net.\*,java.util.\*,java.sql.\*”%>

import属性是唯一可以重复出现的属性：

<%@page import=”java.util.\*” import=”java.net.\*” import=”java.sql.\*”%>

但是，我们一般会使用多个page指令来导入多个包：

<%@ page import=”java.util.\*”%>

<%@ page import=”java.net.\*”%>

<%@ page import=”java.text.\*”%>

2.3　page指令的errorPage和isErrorPage

我们知道，在一个JSP页面出错后，Tomcat会响应给用户错误信息（500页面）！如果你不希望Tomcat给用户输出错误信息，那么可以使用page指令的errorPage来指定错误页！也就是自定义错误页面，例如：<%@page errorPage=”xxx.jsp”%>。这时，在当前JSP页面出现错误时，会请求转发到xxx.jsp页面。

a.jsp

|  |
| --- |
| <%@ page import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ page errorPage=*"b.jsp"* %>  <%  **if**(**true**)  **throw** **new** Exception("哈哈~");  %> |

b.jsp

|  |
| --- |
| <%@ page pageEncoding=*"UTF-8"*%>  <html>  <body>  <h1>出错啦！</h1>  </body>  </html> |

在上面代码中，a.jsp抛出异常后，会请求转发到b.jsp。在浏览器的地址栏中还是a.jsp，因为是请求转发！

而且客户端浏览器收到的响应码为200，表示请求成功！如果希望客户端得到500，那么需要指定b.jsp为错误页面。

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ page isErrorPage=*"true"* %>  <html>  <body>  <h1>出错啦！</h1>  <%=exception.getMessage() %>  </body>  </html> |

　　注意，当isErrorPage为true时，说明当前JSP为错误页面，即专门处理错误的页面。那么这个页面中就可以使用一个内置对象exception了。其他页面是不能使用这个内置对象的！

温馨提示：*IE会在状态码为500时，并且响应正文的长度小于等于512B时不给予显示！而是显示“网站无法显示该页面”字样。这时你只需要添加一些响应内容即可，例如上例中的b.jsp中我给出一些内容，IE就可以正常显示了！*

**2.3.1　web.xml中配置错误页面**

　　不只可以通过JSP的page指令来配置错误页面，还可以在web.xml文件中指定错误页面。这种方式其实与page指令无关，但想来想去还是在这个位置来讲解比较合适！

web.xml

|  |
| --- |
| <error-page>  <error-code>404</error-code>  <location>/error404.jsp</location>  </error-page>  <error-page>  <error-code>500</error-code>  <location>/error500.jsp</location>  </error-page>  <error-page>  <exception-type>java.lang.RuntimeException</exception-type>  <location>/error.jsp</location>  </error-page> |

<error-page>有两种使用方式：

* <error-code>和<location>子元素；
* <exception-type>和<location>子元素；

　　其中<error-code>是指定响应码；<location>指定转发的页面；<exception-type>是指定抛出的异常类型。

在上例中：

* 当出现404时，会跳转到error404.jsp页面；
* 当出现RuntimeException异常时，会跳转到error.jsp页面；
* 当出现非RuntimeException的异常时，会跳转到error500.jsp页面。

　　这种方式会在控制台看到异常信息！而使用page指令时不会在控制台打印异常信息。

2.4　page指令的autFlush和buffer

buffer表示当前JSP的输出流（out隐藏对象）的缓冲区大小，**默认为8kb**。

authFlush表示在out对象的缓冲区满时如果处理！当authFlush为true时，表示缓冲区满时把缓冲区数据输出到客户端；当authFlush为false时，表示缓冲区满时，抛出异常。**authFlush的默认值为true**。

　　这两个属性一般我们也不会去特意设置，都是保留默认值！

2.5　page指令的isELIgnored

　　后面我们会讲解EL表达式语言，page指令的isElIgnored属性表示当前JSP页面是否忽略EL表达式，默认值为false，表示不忽略（即支持）。

2.6　page指令的其他属性

* language：**只能是Java**，这个属性可以看出JSP最初设计时的野心！希望JSP可以转换成其他语言！但是，到现在JSP也只能转换成Java代码；
* info：JSP说明性信息；
* isThreadSafe：默认为false，为true时，JSP生成的Servlet会去实现一个过时的标记接口SingleThreadModel，这时JSP就只能处理单线程的访问；
* session：默认为true，表示当前JSP页面可以使用session对象，如果为false表示当前JSP页面不能使用session对象；
* extends：指定当前JSP页面生成的Servlet的父类；

2.7　<jsp-config>（了解）

在web.xml页面中配置<jsp-config>也可以完成很多page指定的功能！

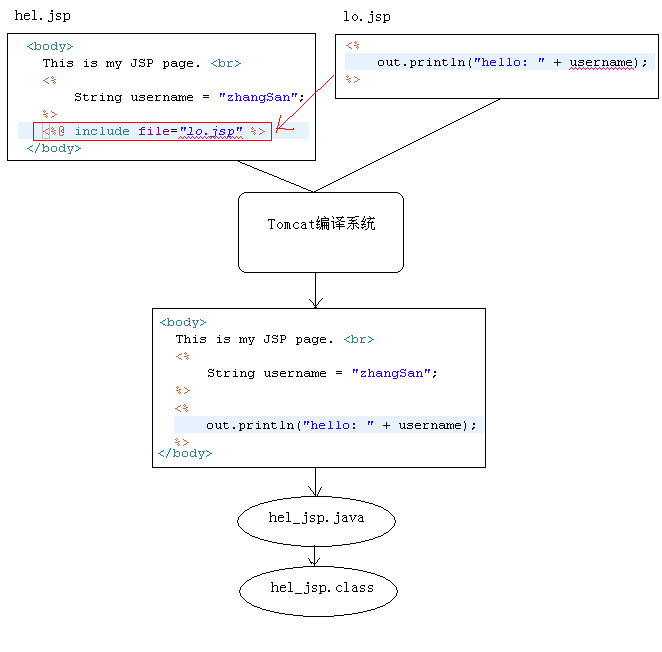
|  |
| --- |
| <jsp-config>  <jsp-property-group>  <url-pattern>\*.jsp</url-pattern>  <el-ignored>true</el-ignored>  <page-encoding>UTF-8</page-encoding>  <scripting-invalid>true</scripting-invalid>  </jsp-property-group>  </jsp-config> |

###### 3　include指令

include指令表示静态包含！即目的是把多个JSP合并成一个JSP文件！

include指令只有一个属性：file，指定要包含的页面，例如：<%@include file=”b.jsp”%>。

静态包含：当hel.jsp页面包含了lo.jsp页面后，在编译hel.jsp页面时，需要把hel.jsp和lo.jsp页面合并成一个文件，然后再编译成Servlet（Java文件）。



很明显，在ol.jsp中在使用username变量，而这个变量在hel.jsp中定义的，所以只有这两个JSP文件合并后才能使用。通过include指定完成对它们的合并！

###### 4　taglib指令

　　这个指令需要在学习了自定义标签后才会使用，现在只能做了了解而已！

在JSP页面中使用第三方的标签库时，需要使用taglib指令来“导包”。例如：

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

其中prefix表示标签的前缀，这个名称可以随便起。uri是由第三方标签库定义的，所以你需要知道第三方定义的uri。

## JSP九大内置对象

###### 1　什么是JSP九大内置对象

在JSP中无需创建就可以使用的9个对象，它们是：

* out（JspWriter）：等同与response.getWriter()，用来向客户端发送文本数据；
* config（ServletConfig）：对应“真身”中的ServletConfig；
* page（当前JSP的真身类型）：当前JSP页面的“this”，即当前对象；
* pageContext（PageContext）：页面上下文对象，它是最后一个没讲的域对象；
* exception（Throwable）：只有在错误页面中可以使用这个对象；
* request（HttpServletRequest）：即HttpServletRequest类的对象；
* response（HttpServletResponse）：即HttpServletResponse类的对象；
* application（ServletContext）：即ServletContext类的对象；
* session（HttpSession）：即HttpSession类的对象，不是每个JSP页面中都可以使用，如果在某个JSP页面中设置<%@page session=”false”%>，说明这个页面不能使用session。

在这9个对象中有很多是极少会被使用的，例如：config、page、exception基本不会使用。

在这9个对象中有两个对象不是每个JSP页面都可以使用的：exception、session。

在这9个对象中有很多前面已经学过的对象：out、request、response、application、session、config。

###### 2　通过“真身”来对照JSP

　　我们知道JSP页面的内容出现在“真身”的\_jspService()方法中，而在\_jspService()方法开头部分已经创建了9大内置对象。

|  |
| --- |
| public void \_jspService(HttpServletRequest request, HttpServletResponse response)  throws java.io.IOException, ServletException {  PageContext pageContext = null;  HttpSession session = null;  ServletContext application = null;  ServletConfig config = null;  JspWriter out = null;  Object page = this;  JspWriter \_jspx\_out = null;  PageContext \_jspx\_page\_context = null;  try {  response.setContentType("text/html;charset=UTF-8");  pageContext = \_jspxFactory.getPageContext(this, request, response,  null, true, 8192, true);  \_jspx\_page\_context = pageContext;  application = pageContext.getServletContext();  config = pageContext.getServletConfig();  session = pageContext.getSession();  out = pageContext.getOut();  \_jspx\_out = out;  **从这里开始，才是JSP页面的内容**  }… |

###### 3　pageContext对象

　　在JavaWeb中一共四个域对象，其中Servlet中可以使用的是request、session、application三个对象，而在JSP中可以使用pageContext、request、session、application四个域对象。

pageContext 对象是PageContext类型，它的主要功能有：

* 域对象功能；
* 代理其它域对象功能；
* 获取其他内置对象；

3.1　域对象功能

　　pageContext也是域对象，它的范围是当前页面。它的范围也是四个域对象中最小的！

* void setAttribute(String name, Object value)；
* Object getAttrbiute(String name, Object value)；
* void removeAttribute(String name, Object value)；

3.2　代理其它域对象功能

还可以使用pageContext来代理其它3个域对象的功能，也就是说可以使用pageContext向request、session、application对象中存取数据，例如：

|  |
| --- |
| pageContext.setAttribute("x", "X");  pageContext.setAttribute("x", "XX", PageContext.REQUEST\_SCOPE);  pageContext.setAttribute("x", "XXX", PageContext.SESSION\_SCOPE);  pageContext.setAttribute("x", "XXXX", PageContext.APPLICATION\_SCOPE); |

* void setAttribute(String name, Object value, int scope)：在指定范围中添加数据；
* Object getAttribute(String name, int scope)：获取指定范围的数据；
* void removeAttribute(String name, int scope)：移除指定范围的数据；
* Object findAttribute(String name)：依次在page、request、session、application范围查找名称为name的数据，如果找到就停止查找。这说明在这个范围内有相同名称的数据，那么page范围的优先级最高！

3.3　获取其他内置对象

一个pageContext对象等于所有内置对象，即1个当9个。这是因为可以使用pageContext对象获取其它8个内置对象：

* JspWriter getOut()：获取out内置对象；
* ServletConfig getServletConfig()：获取config内置对象；
* Object getPage()：获取page内置对象；
* ServletRequest getRequest()：获取request内置对象；
* ServletResponse getResponse()：获取response内置对象；
* HttpSession getSession()：获取session内置对象；
* ServletContext getServletContext()：获取application内置对象；
* Exception getException()：获取exception内置对象；

## JSP动作标签

###### 1　JSP动作标签概述

动作标签的作用是用来简化Java脚本的！

JSP动作标签是JavaWeb内置的动作标签，它们是已经定义好的动作标签，我们可以拿来直接使用。

如果JSP动作标签不够用时，还可以使用自定义标签（今天不讲）。JavaWeb一共提供了20个JSP动作标签，但有很多基本没有用，这里只介绍一些有坐标的动作标签。

JSP动作标签的格式：<jsp:标签名 …>

###### 2　<jsp:include>

<jsp:include>标签的作用是用来包含其它JSP页面的！你可能会说，前面已经学习了include指令了，它们是否相同呢？虽然它们都是用来包含其它JSP页面的，但它们的实现的级别是不同的！

include指令是在编译级别完成的包含，即把当前JSP和被包含的JSP合并成一个JSP，然后再编译成一个Servlet。

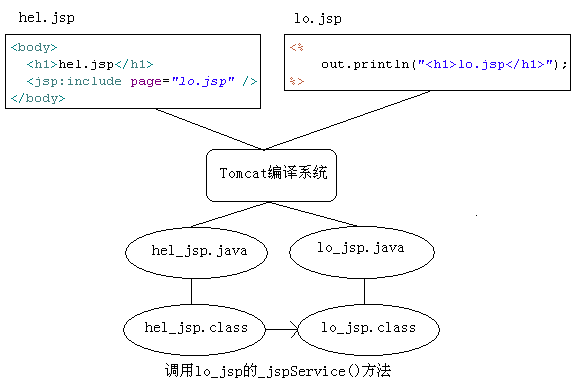
include动作标签是在运行级别完成的包含，即当前JSP和被包含的JSP都会各自生成Servlet，然后在执行当前JSP的Servlet时完成包含另一个JSP的Servlet。它与RequestDispatcher的include()方法是相同的！

hel.jsp

|  |
| --- |
| <body>  <h1>hel.jsp</h1>  <jsp:include page=*"lo.jsp"* />  </body> |

lo.jsp

|  |
| --- |
| <%  out.println("<h1>lo.jsp</h1>");  %> |



　　其实<jsp:include>在“真身”中不过是一句方法调用，即调用另一个Servlet而已。

###### 3　<jsp:forward>

forward标签的作用是请求转发！forward标签的作用与RequestDispatcher#forward()方法相同。

hel.jsp

|  |
| --- |
|  |

lo.jsp

|  |
| --- |
| <%  out.println("<h1>lo.jsp</h1>");  %> |

注意，最后客户端只能看到lo.jsp的输出，而看不到hel.jsp的内容。也就是说在hel.jsp中的<h1>hel.jsp</h1>是不会发送到客户端的。<jsp:forward>的作用是“别在显示我，去显示它吧！”。

###### 4　<jsp:param>

还可以在<jsp:include>和<jsp:forward>标签中使用<jsp:param>子标签，它是用来传递参数的。下面用<jsp:include>来举例说明<jsp:param>的使用。

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>a.jsp</title>  </head>    <body>  <h1>a.jsp</h1>  <hr/>  <jsp:include page=*"/b.jsp"*>  <jsp:param value=*"zhangSan"* name=*"username"*/>  </jsp:include>  </body>  </html> |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>b.jsp</title>  </head>    <body>  <h1>b.jsp</h1>  <hr/>  <%  String username = request.getParameter("username");  out.print("你好：" + username);  %>  </body>  </html> |

## JavaBean

###### 1　JavaBean概述

1.1　什么是JavaBean

JavaBean是一种规范，也就是对类的要求。它要求Java类的成员变量提供getter/setter方法，这样的成员变量被称之为JavaBean属性。

JavaBean还要求类必须提供仅有的无参构造器，例如：public User() {…}

User.java

|  |
| --- |
| **package** cn.itcast.domain;  **public** **class** User {  **private** String username;  **private** String password;  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPassword() {  **return** password;  }  **public** **void** setPassword(String password) {  **this**.password = password;  }  } |

1.2　JavaBean属性

JavaBean属性是具有getter/setter方法的成员变量。

* 也可以只提供getter方法，这样的属性叫只读属性；
* 也可以只提供setter方法，这样的属性叫只写属性；
* 如果属性类型为boolean类型，那么读方法的格式可以是get或is。例如名为abc的boolean类型的属性，它的读方法可以是getAbc()，也可以是isAbc()；

JavaBean属性名要求：前两个字母要么都大写，要么都小写：

|  |
| --- |
| **public** **class** User {  **private** String iD;  **private** String ID;  **private** String qQ;  **private** String QQ;  …  } |

JavaBean可能存在属性，但不存在这个成员变量，例如：

|  |
| --- |
| **public** **class** User {  **public** String getUsername() {  **return** "zhangSan";  }  } |

　　上例中User类有一个名为username的只读属性！但User类并没有username这个成员变量！

还可以并变态一点：

|  |
| --- |
| **public** **class** User {  **private** String hello;    **public** String getUsername() {  **return** hello;  }    **public** **void** setUsername(String username) {  **this**.hello = username;  }  } |

　　上例中User类中有一个名为username的属性，它是可读可写的属性！而Use类的成员变量名为hello！也就是说JavaBean的属性名取决与方法名称，而不是成员变量的名称。但通常没有人做这么变态的事情。

###### 2　内省（了解）

内省的目标是得到JavaBean属性的读、写方法的反射对象，通过反射对JavaBean属性进行操作的一组API。例如User类有名为username的JavaBean属性，通过两个Method对象（一个是getUsenrmae()，一个是setUsername()）来操作User对象。

如果你还不能理解内省是什么，那么我们通过一个问题来了解内省的作用。现在我们有一个Map，内容如下：

|  |
| --- |
| Map<String,String> map = **new** HashMap<String,String>();  map.put("username", "admin");  map.put("password", "admin123"); |
| **public** **class** User {  **private** String username;  **private** String password;  **public** User(String username, String password) {  **this**.username = username;  **this**.password = password;  }  **public** User() {  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPassword() {  **return** password;  }  **public** **void** setPassword(String password) {  **this**.password = password;  }  **public** String toString() {  **return** "User [username=" + username + ", password=" + password + "]";  }  } |

现在需要把map的数据封装到一个User对象中！User类有两个JavaBean属性，一个叫username，另一个叫password。

你可能想到的是反射，通过map的key来查找User类的Field！这么做是没有问题的，但我们要知道类的成员变量是私有的，虽然也可以通过反射去访问类的私有的成员变量，但我们也要清楚反射访问私有的东西是有“危险”的，所以还是建议通过getUsername和setUsername来访问JavaBean属性。

2.1　内省之获取BeanInfo

我们这里不想去对JavaBean规范做过多的介绍，所以也就不在多介绍BeanInfo的“出身”了。你只需要知道如何得到它，以及BeanInfo有什么。

通过java.beans.Introspector的getBeanInfo()方法来获取java.beans.BeanInfo实例。

|  |
| --- |
| BeanInfo beanInfo = Introspector.*getBeanInfo*(User.**class**); |

2.2　得到所有属性描述符（PropertyDescriptor）

通过BeanInfo可以得到这个类的所有JavaBean属性的PropertyDescriptor对象。然后就可以通过PropertyDescriptor对象得到这个属性的getter/setter方法的Method对象了。

|  |
| --- |
| PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors(); |

每个PropertyDescriptor对象对应一个JavaBean属性：

* String getName()：获取JavaBean属性名称；
* Method getReadMethod：获取属性的读方法；
* Method getWriteMethod：获取属性的写方法。

2.3　完成Map数据封装到User对象中

|  |
| --- |
| **public** **void** fun1() **throws** Exception {  Map<String,String> map = **new** HashMap<String,String>();  map.put("username", "admin");  map.put("password", "admin123");    BeanInfo beanInfo = Introspector.*getBeanInfo*(User.**class**);    PropertyDescriptor[] pds = beanInfo.getPropertyDescriptors();    User user = **new** User();  **for**(PropertyDescriptor pd : pds) {  String name = pd.getName();  String value = map.get(name);  **if**(value != **null**) {  Method writeMethod = pd.getWriteMethod();  writeMethod.invoke(user, value);  }  }    System.*out*.println(user);  } |

###### 3　commons-beanutils

提到内省，不能不提commons-beanutils这个工具。它底层使用了内省，对内省进行了大量的简化！

使用beanutils需要的jar包：

* commons-beanutils.jar；
* commons-logging.jar；

3.1　设置JavaBean属性

|  |
| --- |
| User user = **new** User();    BeanUtils.*setProperty*(user, "username", "admin");  BeanUtils.*setProperty*(user, "password", "admin123");    System.*out*.println(user); |

3.2　获取JavaBean属性

|  |
| --- |
| User user = **new** User("admin", "admin123");    String username = BeanUtils.*getProperty*(user, "username");  String password = BeanUtils.*getProperty*(user, "password");    System.*out*.println("username=" + username + ", password=" + password); |

3.3　封装Map数据到JavaBean对象中

|  |
| --- |
| Map<String,String> map = **new** HashMap<String,String>();  map.put("username", "admin");  map.put("password", "admin123");    User user = **new** User();  BeanUtils.*populate*(user, map);    System.*out*.println(user); |

###### 4　JSP与JavaBean相关的动作标签

在JSP中与JavaBean相关的标签有：

* <jsp:useBean>：创建JavaBean对象；
* <jsp:setProperty>：设置JavaBean属性；
* <jsp:getProperty>：获取JavaBean属性；

我们需要先创建一个JavaBean类：

User.java

|  |
| --- |
| **package** cn.itcast.domain;  **public** **class** User {  **private** String username;  **private** String password;  **public** User(String username, String password) {  **this**.username = username;  **this**.password = password;  }  **public** User() {  }  **public** String getUsername() {  **return** username;  }  **public** **void** setUsername(String username) {  **this**.username = username;  }  **public** String getPassword() {  **return** password;  }  **public** **void** setPassword(String password) {  **this**.password = password;  }  **public** String toString() {  **return** "User [username=" + username + ", password=" + password + "]";  }  } |

4.1　<jsp:useBean>

<jsp:useBean>标签的作用是创建JavaBean对象：

* 在当前JSP页面创建JavaBean对象；
* 把创建的JavaBean对象保存到域对象中；

|  |
| --- |
| <jsp:useBean id=*"user1"* class=*"cn.itcast.domain.User"* /> |

上面代码表示在当前JSP页面中创建User类型的对象，并且把它保存到page域中了。下面我们把<jsp:useBean>标签翻译成Java代码：

|  |
| --- |
| <%  cn.itcast.domain.User user1 = **new** cn.itcast.domain.User();  pageContext.setAttribute("user1", user1);  %> |

这说明我们可以在JSP页面中完成下面的操作：

|  |
| --- |
| <jsp:useBean id=*"user1"* class=*"cn.itcast.domain.User"* />  <%=user1 %>  <%  out.println(pageContext.getAttribute("user1"));  %> |

<jsp:useBean>标签默认是把JavaBean对象保存到page域，还可以通过scope标签属性来指定保存的范围：

|  |
| --- |
| <jsp:useBean id=*"user1"* class=*"cn.itcast.domain.User"* scope=*"page"*/>  <jsp:useBean id=*"user2"* class=*"cn.itcast.domain.User"* scope=*"request"*/>  <jsp:useBean id=*"user3"* class=*"cn.itcast.domain.User"* scope=*"session"*/>  <jsp:useBean id=*"user4"* class=*"cn.itcast.domain.User"* scope=*"applicatioin"*/> |

<jsp:useBean>标签其实不一定会创建对象！！！其实它会先在指定范围中查找这个对象，如果对象不存在才会创建，我们需要重新对它进行翻译：

|  |
| --- |
| <jsp:useBean id=*"user4"* class=*"cn.itcast.domain.User"* scope=*"applicatioin"*/> |
| <%  cn.itcast.domain.User user4 = (cn.itcast.domain.User)application.getAttribute("user4");  **if**(user4 == **null**) {  user4 = **new** cn.itcast.domain.User();  application.setAttribute("user4", user4);  }  %> |

4.2　<jsp:setProperty>和<jsp:getProperty>

<jsp:setProperty>标签的作用是给JavaBean设置属性值，而<jsp:getProperty>是用来获取属性值。在使用它们之前需要先创建JavaBean：

|  |
| --- |
| <jsp:useBean id=*"user1"* class=*"cn.itcast.domain.User"* />  <jsp:setProperty property=*"username"* name=*"user1"* value=*"admin"*/>  <jsp:setProperty property=*"password"* name=*"user1"* value=*"admin123"*/>  用户名：<jsp:getProperty property=*"username"* name=*"user1"*/><br/>  密　码：<jsp:getProperty property=*"password"* name=*"user1"*/><br/> |

## EL（表达式语言）

###### 1　EL概述

1.1　EL的作用

JSP2.0要把html和css分离、要把html和javascript分离、要把Java脚本替换成标签。标签的好处是非Java人员都可以使用。

JSP2.0 – 纯标签页面，即：不包含<% … %>、<%! … %>，以及<%= … %>

EL（Expression Language）是一门表达式语言，它对应<%=…%>。我们知道在JSP中，表达式会被输出，所以EL表达式也会被输出。

1.2　EL的格式

格式：${…}

例如：${1 + 2}

1.3　关闭EL

如果希望整个JSP忽略EL表达式，需要在page指令中指定isELIgnored=”true”。

如果希望忽略某个EL表达式，可以在EL表达式之前添加“\”，例如：\${1 + 2}。

1.4　EL运算符

|  |  |  |  |
| --- | --- | --- | --- |
| 运算符 | 说明 | 范例 | 结果 |
| + | 加 | ${17+5} | 22 |
| - | 减 | ${17-5} | 12 |
| \* | 乘 | ${17\*5} | 85 |
| /或div | 除 | ${17/5}或${17 div 5} | 3 |
| %或mod | 取余 | ${17%5}或${17 mod 5} | 2 |
| ==或eq | 等于 | ${5==5}或${5 eq 5} | true |
| !=或ne | 不等于 | ${5!=5}或${5 ne 5} | false |
| <或lt | 小于 | ${3<5}或${3 lt 5} | true |
| >或gt | 大于 | ${3>5}或${3 gt 5} | false |
| <=或le | 小于等于 | ${3<=5}或${3 le 5} | true |
| >=或ge | 大于等于 | ${3>=5}或${3 ge 5} | false |
| &&或and | 并且 | ${true&&false}或${true and false} | false |
| !或not | 非 | ${!true}或${not true} | false |
| ||或or | 或者 | ${true||false}或${true or false} | true |
| empty | 是否为空 | ${empty “”}，可以判断字符串、数据、集合的长度是否为0，为0返回true。empty还可以与not或!一起使用。${not empty “”} | true |

1.5　EL不显示null

　　当EL表达式的值为null时，会在页面上显示空白，即什么都不显示。

###### 2　EL表达式格式

先来了解一下EL表达式的格式！现在还不能演示它，因为需要学习了EL11个内置对象后才方便显示它。

* 操作List和数组：${list[0]}、${arr[0]}；
* 操作bean的属性：${person.name}、${person[‘name’]}，对应person.getName()方法；
* 操作Map的值：${map.key}、${map[‘key’]}，对应map.get(key)。

###### 3　EL内置对象

EL一共11个内置对象，无需创建即可以使用。这11个内置对象中有10个是Map类型的，最后一个是pageContext对象。

* pageScope
* requestScope
* sessionScope
* applicationScope
* param；
* paramValues；
* header；
* headerValues；
* initParam；
* cookie；
* pageContext；

3.1　域相关内置对象（重点）

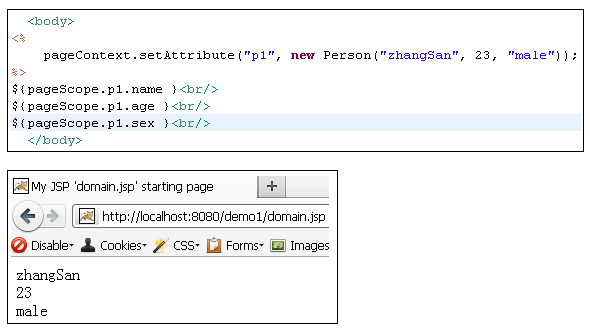
域内置对象一共有四个：

* pageScope：${pageScope.name}等同与pageContext.getAttribute(“name”)；
* requestScope：${requestScope.name}等同与request.getAttribute(“name”)；
* sessionScoep： ${sessionScope.name}等同与session.getAttribute(“name”)；
* applicationScope：${applicationScope.name}等同与application.getAttribute(“name”)；

如果在域中保存的是JavaBean对象，那么可以使用EL来访问JavaBean属性。因为EL只做读取操作，所以JavaBean一定要提供get方法，而set方法没有要求。

Person.java

|  |
| --- |
| **public** **class** Person {  **private** String name;  **private** **int** age;  **private** String sex;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getAge() {  **return** age;  }  **public** **void** setAge(**int** age) {  **this**.age = age;  }  **public** String getSex() {  **return** sex;  }  **public** **void** setSex(String sex) {  **this**.sex = sex;  }  } |

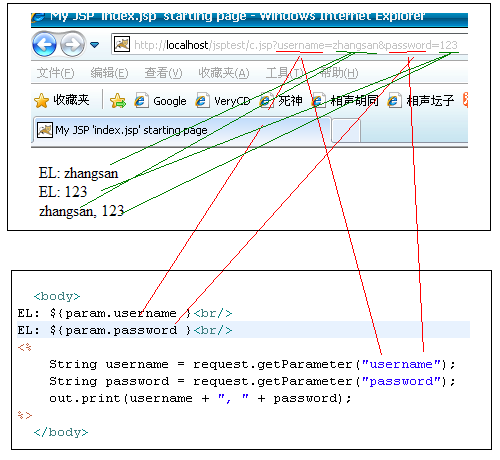


　　全域查找：${person}表示依次在pageScope、requesScopet、sessionScope、appliationScope四个域中查找名字为person的属性。

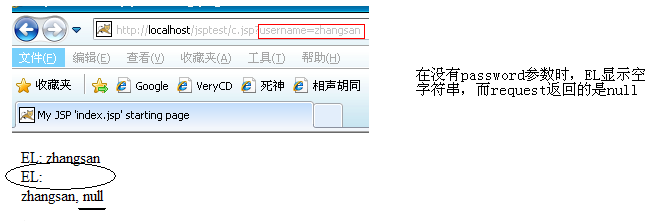
3.2　请求参数相关内置对象

　　param和paramValues这两个内置对象是用来获取请求参数的。

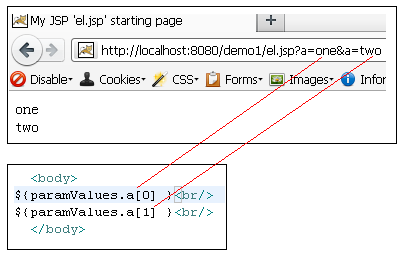
* param：Map<String,String>类型，param对象可以用来获取参数，与request.getParameter()方法相同。



注意，在使用EL获取参数时，如果参数不存在，返回的是空字符串，而不是null。这一点与使用request.getParameter()方法是不同的。



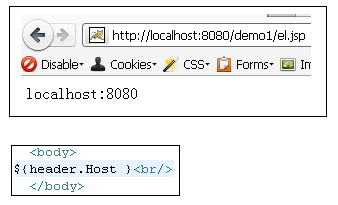
* paramValues：paramValues是Map<String, String[]>类型，当一个参数名，对应多个参数值时可以使用它。



3.3　请求头相关内置对象

header和headerValues是与请求头相关的内置对象：

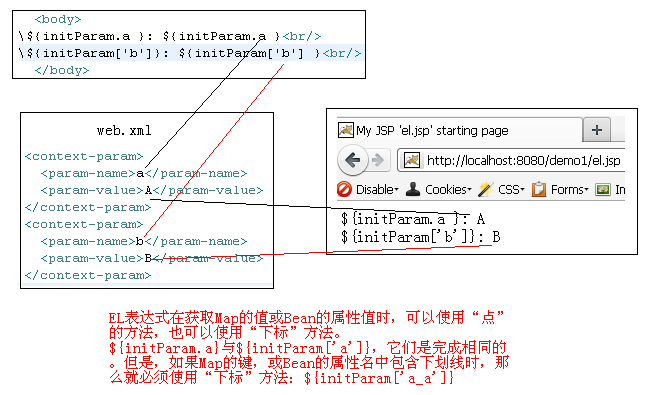
* header： Map<String,String>类型，用来获取请求头。



* headerValues：headerValues是Map<String,String[]>类型。当一个请求头名称，对应多个值时，使用该对象，这里就不在赘述。

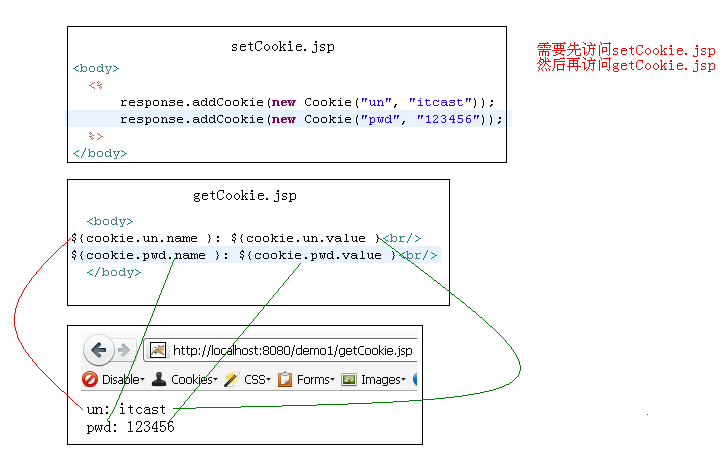
3.4　应用初始化参数相关内置对象

* initParam：initParam是Map<String,String>类型。它对应web.xml文件中的<context-param>参数。



3.5　Cookie相关内置对象

* cookie：cookie是Map<String,Cookie>类型，其中key是Cookie的名字，而值是Cookie对象本身。



3.6　pageContext对象

pageContext：pageContext是PageContext类型！可以使用pageContext对象调用getXXX()方法，例如pageContext.getRequest()，可以${pageContext.request}。也就是读取JavaBean属性！！！

|  |  |
| --- | --- |
| **EL表达式** | **说明** |
| ${pageContext.request.queryString} | pageContext.getRequest().getQueryString(); |
| ${pageContext.request.requestURL} | pageContext.getRequest().getRequestURL(); |
| ${pageContext.request.contextPath} | pageContext.getRequest().getContextPath(); |
| ${pageContext.request.method} | pageContext.getRequest().getMethod(); |
| ${pageContext.request.protocol} | pageContext.getRequest().getProtocol(); |
| ${pageContext.request.remoteUser} | pageContext.getRequest().getRemoteUser(); |
| ${pageContext.request.remoteAddr} | pageContext.getRequest().getRemoteAddr(); |
| ${pageContext.session.new} | pageContext.getSession().isNew(); |
| ${pageContext.session.id} | pageContext.getSession().getId(); |
| ${pageContext.servletContext.serverInfo} | pageContext.getServletContext().getServerInfo(); |

## EL函数库

###### 1　什么EL函数库

　　EL函数库是由第三方对EL的扩展，我们现在学习的EL函数库是由JSTL添加的。JSTL明天再学！

EL函数库就是定义一些有**返回值**的**静态方法**。然后通过EL语言来调用它们！当然，不只是JSTL可以定义EL函数库，我们也可以自定义EL函数库。

　　EL函数库中包含了很多对字符串的操作方法，以及对集合对象的操作。例如：${fn:length(“abc”)}会输出3，即字符串的长度。

###### 2　导入函数库

　　因为是第三方的东西，所以需要导入。导入需要使用taglib指令！

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

###### 3　EL函数库介绍

* String toUpperCase(String input)：
* String toLowerCase(String input)：
* int indexOf(String input, String substring)：
* boolean contains(String input, String substring)：
* boolean containsIgnoreCase(String input, String substring)：
* boolean startsWith(String input, String substring)：
* boolean endsWith(String input, String substring)：
* String substring(String input, int beginIndex, int endIndex)：
* String substringAfter(String input, String substring)：hello-world, “-“
* substringBefore(String input, String substring)：hello-world, “-“
* String escapeXml(String input)：把字符串的“>”、“<”。。。转义了！
* String trim(String input)：
* String replace(String input, String substringBefore, String substringAfter)：
* String[] split(String input, String delimiters)：
* int length(Object obj)：可以获取字符串、数组、各种集合的长度！
* String join(String array[], String separator)：

|  |
| --- |
| <%@taglib prefix=*"fn"* uri=*"http://java.sun.com/jsp/jstl/functions"* %>  …  String[] strs = {"a", "b","c"};  List list = **new** ArrayList();  list.add("a");  pageContext.setAttribute("arr", strs);  pageContext.setAttribute("list", list);  %>  ${fn:length(arr) }<br/><!--3-->  ${fn:length(list) }<br/><!--1-->  ${fn:toLowerCase("Hello") }<br/> <!-- hello -->  ${fn:toUpperCase("Hello") }<br/> <!-- HELLO -->  ${fn:contains("abc", "a")}<br/><!-- true -->  ${fn:containsIgnoreCase("abc", "Ab")}<br/><!-- true -->  ${fn:contains(arr, "a")}<br/><!-- true -->  ${fn:containsIgnoreCase(list, "A")}<br/><!-- true -->  ${fn:endsWith("Hello.java", ".java")}<br/><!-- true -->  ${fn:startsWith("Hello.java", "Hell")}<br/><!-- true -->  ${fn:indexOf("Hello-World", "-")}<br/><!-- 5 -->  ${fn:join(arr, ";")}<br/><!-- a;b;c -->  ${fn:replace("Hello-World", "-", "+")}<br/><!-- Hello+World -->  ${fn:join(fn:split("a;b;c;", ";"), "-")}<br/><!-- a-b-c -->  ${fn:substring("0123456789", 6, 9)}<br/><!-- 678 -->  ${fn:substring("0123456789", 5, -1)}<br/><!-- 56789 -->  ${fn:substringAfter("Hello-World", "-")}<br/><!-- World -->  ${fn:substringBefore("Hello-World", "-")}<br/><!-- Hello -->  ${fn:trim(" a b c ")}<br/><!-- a b c -->  ${fn:escapeXml("<html></html>")}<br/> <!-- <html></html> --> |

###### 4　自定义EL函数库

* 写一个类，写一个有返回值的静态方法；
* 编写itcast.tld文件，可以参数fn.tld文件来写，把itcast.tld文件放到/WEB-INF目录下；
* 在页面中添加taglib指令，导入自定义标签库。

ItcastFuncations.java

|  |
| --- |
| **package** cn.itcast.el.funcations;  **public** **class** ItcastFuncations {  **public** **static** String test() {  **return** "传智播客自定义EL函数库测试";  }  } |

itcast.tld（放到classes下）

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <taglib xmlns=*"http://java.sun.com/xml/ns/j2ee"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary\_2\_0.xsd"*  version=*"2.0"*>    <tlib-version>1.0</tlib-version>  <short-name>itcast</short-name>  <uri>http://www.itcast.cn/jsp/functions</uri>  <function>  <name>test</name>  <function-class>cn.itcast.el.funcations.ItcastFuncations</function-class>  <function-signature>String test()</function-signature>  </function>  </taglib> |

index.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib prefix=*"itcast"* uri=*"/WEB-INF/itcast.tld"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <body>  <h1>${itcast:test() }</h1>  </body>  </html> |