目录

[环境搭建 2](#_Toc487640326)

[基础工具配置 2](#_Toc487640327)

[运行环境Tomcat配置及改端口 2](#_Toc487640328)

[jsp基础语法及应用 2](#_Toc487640329)

[Tomcat文件生成位置 2](#_Toc487640330)

[基础方法 2](#_Toc487640331)

[<%-- java注释 --%> 2](#_Toc487640332)

[<% java代码块 %> 2](#_Toc487640333)

[<%! java申明语句块 %> 2](#_Toc487640334)

[<%=java表达式块 %> 3](#_Toc487640335)

[session与pageContext的设置与调取 3](#_Toc487640336)

[import引用类库 3](#_Toc487640337)

[errorPage错误跳转页面 500错误 3](#_Toc487640338)

[Include 静态联编与动态联编 4](#_Toc487640339)

[Action 七大动作，不常用 4](#_Toc487640340)

[Request方法汇总 4](#_Toc487640341)

[EL访问 4](#_Toc487640342)

[EL从四大域中取数据 4](#_Toc487640343)

[EL访问Bean的属性 也就是构造函数 4](#_Toc487640344)

[EL访问数组 5](#_Toc487640345)

[EL访问List 5](#_Toc487640346)

[EL访问Map 5](#_Toc487640347)

[EL中的运算符 5](#_Toc487640348)

[EL内置对象 5](#_Toc487640349)

[EL注册（pageContext延展） 6](#_Toc487640350)

[EL自定义函数（有实例） 7](#_Toc487640351)

[EL使用JSTL函数库 8](#_Toc487640352)

[EL总结 8](#_Toc487640353)

[自定义标签（整套自定义标签在一个实例） 8](#_Toc487640354)

[基本用法 8](#_Toc487640355)

[标签体 10](#_Toc487640356)

[带属性 10](#_Toc487640357)

[List的forEachList 12](#_Toc487640358)

# 环境搭建

## 基础工具配置

解压eclipse 配置JDK

## 运行环境Tomcat配置及改端口

联网状态下 打开eclipse windows

preferences -- Server -- Runtime Environments

Add -- apache -- Apache Tomcat v7.0 -- Next -- Download and Install

或者直接下载

端口修改 这里的port里的内容改掉

Conf server.xml文件  
<Connector port="8080"

# jsp基础语法及应用

## Tomcat文件生成位置

生成的jsp原文件

F:\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\01jsp

访问后jsp转成 java文件，完了再转成class文件

F:\workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\01jsp\org\apache\jsp

备注：在工作空间下 01jsp为工程名

## 基础方法

<%-- java注释 --%> 当然//这些同样适用

<% java代码块 %>

备注：<% %>中 由于jsp脚本是被翻译到了Servlet的 \_jspServlet()方法中，所以jsp小脚本中是不能出现以下内容的：

1. 声明的变量不能加权限控制符 例如public private
2. 不能定义方法
3. 不能定义静态语句块 例如static

<%! java申明语句块 %>

备注：声明语句块由于是加在 类里边， \_jspServlet()方法外边，所以可以定义公共的变量和方法

### <%=java表达式块 %>

备注：结束的地方不加;

## session与pageContext的设置与调取

<%

session.setAttribute("user","aac");

String user=(String)session.getAttribute("user");

%>

<%=user %>

session进阶： 太复杂，其实没必要

若向session域中存放数据，则getSession(true),即getSession()

若从session域中读取数据，则getSession(false)

<%@ page session="false" %> 强制session不调用公共的

HttpSession s= request.getSession(false);

if(s!=null){

String user=(String)s.getAttribute("user");

out.print("user"+user);

}

后边就需要代码去创建session实例;

<%

pageContext.setAttribute("xxx","asd");

String pa=(String)pageContext.getAttribute("xxx");

%>

<%=pa%>

备注：pageContext是只有当前页有效，一般用不到

## import引用类库

<%@ page import=*"java.util.Date"* %>

如何有多个，在之前加个, 或者再写一个

## errorPage错误跳转页面 500错误

<%@ page errorPage=*"/error.jsp"* %>

在错误页面 设置 <%@ page isErrorPage=*"true"* %>

可以通过 <%=exception.getMessage() %>来查看错误

## Include 静态联编与动态联编

<%@ include file="/include/head.jsp" %>

备注：公共文件 与母体之间的 变量 参数等共享。 即使eclipse里报错，后边也是可以的

<jsp:include page="/include/head.jsp"></jsp:include>

备注：公共文件与母体之间生成的是两个文件，里边的任何变量参数是不是共享的

## Action 七大动作，不常用

<jsp:forward page="/include/head.jsp"></jsp:forward>

## Request方法汇总

request.getRemoteAddr(); 获取当前网站IP

request.getContextPath() 获取当前项目名称

# EL访问

## EL从四大域中取数据

${变量}

这个变量的定义只能来自于四大域，否则调取不到

pageContext.setAttribute(“”,””)

request.setAttribute(“”,””)

session.setAttribute(“”,””)

application.setAttribute(“”,””)

如果定义的一个变量四大域里边都有，会从小到大去找，找到了就不继续找下去了

pageContext<request<session<application

如何一个个去找，效率太低，可在后边加Scope直接去某个域里找

${sessionScope.x}

## EL访问Bean的属性 也就是构造函数

bean里边变量要做get set处理 完了无参构造器和带参构造器**super**()

${x.name} ${x.age}

${x["name"]} ${x["age"]}

${x['name']} ${x['age']}

若访问的是null对象或属性，不会报错，直接什么都不显示

如果有多层就 ${x.school.sname}

## EL访问数组

<%

String[] names={"张三","李四","王五"};

pageContext.setAttribute("x", names);

%>

${x[1]}

## EL访问List

import="java.util.\*"

List<String> names= new ArrayList<String>();

names.add("张三");

names.add("李四");

names.add("王五");

pageContext.setAttribute("names", names);

%>

${names[0]}

## EL访问Map

<%

Map<String,Object> map=new HashMap<String,Object>();

map.put("school", "清华大学");

map.put("age", 21);

pageContext.setAttribute("x", map);

%>

${x.school}

## EL中的运算符

[++](http://www.baidu.com/link?url=H_n1pJCyxRZlj6MJAZx8D-BerIzVxcdNPIB0GYdspGqs-pOFSf-VMf1dgdtZeon2aIRTYWmThVgORcu9hyhOCDfzGwBp_k74ipKyFfwkHOa) -- 不能用

empty 对于

没有定义的变量

值为null

值为””

没有元素的数组或集合

的运算结果为ture

${empty name}

## EL内置对象

除了常用的四大域，总共有11大内置对象

常用的：

除了pageContext外，其它10个内置对象都是java.util.Map类型

param 获取请求中的指定参数值

${param.a} 获取<input type="test" name="a">

paramValues 获取请求中的指定参数值

${paramValues.name[0]} 这种格式要加[i] 比较适用于复选按钮

initParam 获取初始化数据里的参数

${initParam.a }

WEB-INF下web.xml

<context-param>

<param-name>a</param-name>

<param-value>你好</param-value>

</context-param>

## EL注册（pageContext延展）

${pageContext.request.contextPath } web应用的根

Jsp页面的写法

<form action="${pageContext.request.contextPath }/registerServer" method="POST">

姓名：<input type="test" name="name"><br>

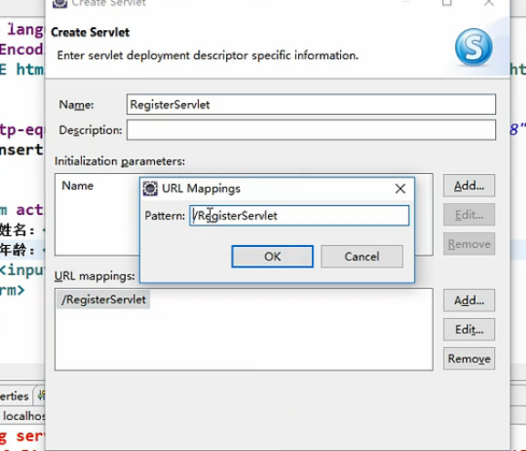
年龄：<input type="text" name="age"><br>

<input type="submit" value="提交">

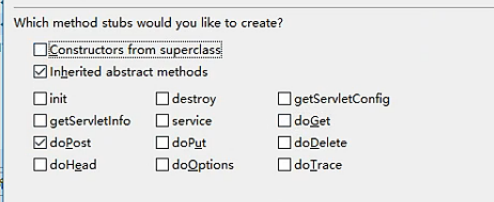
</form>

建个servlet

这里修改访问地址



这里修改访问方式



以下是动作

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

System.out.println("sssssssssssss");

}

## EL自定义函数（有实例）

1. 创建class

package com.lonhwin.functions;

public class ELfunctions {

//将字符串小写变大写

public static String lowerToUpper(String source) {

return source.toUpperCase();

}

public static String reA(String a){

return "你好："+a;

}

}

1. 将里边的方法注册

在WEB-INF里边创建一个xml,后边后缀改成.tld

这个.tld的文件格式需要约束的，即需要配置文件头部。这个头部约束可以从以下文件中进行复制

在tomcat安装目录下：tomcat7\webapps\examples\WEB-INF\jsp2 文件名：jsp2-example-taglib.tld

下边附基本格式

<?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"*>

<!-- 定义标签库 -->

<description>描述可删</description>

<tlib-version>1.0</tlib-version>

<short-name>myFt</short-name>

<uri>xman</uri>

<!-- 注册函数 -->

<function>

<description>描述可删</description>

<name>myLowerToUpper</name>

<function-class>com.lonhwin.functions.ELfunctions</function-class>

<function-signature>java.lang.String lowerToUpper( java.lang.String )</function-signature>

</function>

<function>

<description>描述可删</description>

<name>reA</name>

<function-class>com.lonhwin.functions.ELfunctions</function-class>

<function-signature>java.lang.String reA( java.lang.String )</function-signature>

</function>

</taglib>

1. 在jsp页面的调用

${myFt:myLowerToUpper("abcd")}

如果是动态变量，必须弄到四大域中

<%

String a="jkl";

pageContext.setAttribute("a", a);

%>

${myFt:reA(a)}

## EL使用JSTL函数库

将jstl-1.2.jar 和 standard-1.1.2.jar 粘贴进 WEB-INF下的lib里

## EL总结

EL不能出现在java代码块，表达式块等jsp的动态代码部分

EL只能从pageContext request session application 四大域属性空间中获取数据

EL 不会跳出空指针异常。若访问一个null对象的属性，则什么也不显示

EL不具有对字符串进行处理的能力，就连简单的字符串连接也不行

# 自定义标签（整套自定义标签在一个实例）

## 基本用法

1. 创建class



这里继承了超类 javax.servlet.jsp.tagext.SimpleTagSupport

**package** com.lonhwin.tag;

**import** java.io.IOException;

**import** javax.servlet.ServletRequest;

**import** javax.servlet.jsp.JspException;

**import** javax.servlet.jsp.JspWriter;

**import** javax.servlet.jsp.PageContext;

**import** javax.servlet.jsp.tagext.SimpleTagSupport;

//定义标签处理器：获取客户端 IP

**public** **class** ipTag **extends** SimpleTagSupport {

@Override

**public** **void** doTag() **throws** JspException, IOException {

//获取pageContext对象

PageContext pc= (PageContext)**this**.getJspContext();

//获取请求对象

ServletRequest request =pc.getRequest();

//获取客户端IP

String clientIp=request.getRemoteAddr();

//获取标准输出流

JspWriter out=pc.getOut();

//将客户端IP写入到输出流中

out.print(clientIp);

}

}

附：以下方法可以 调取jsp里的 out.print 注册标签会比较复杂，如果是注册方法就一个return就解决了

PageContext pc= (PageContext)**this**.getJspContext();

//获取标准输出流

JspWriter out=pc.getOut();

//将客户端IP写入到输出流中

out.print("你好");

2.将里边的方法注册

在WEB-INF里边创建一个xml,后边后缀改成.tld

这里边的设置参照自定义函数，但是这里标签为

<!-- 注册标签 -->

<tag>

<description>描述可删</description>

<name>ipTag</name>

<tag-class>com.lonhwin.tag.ipTag</tag-class>

<body-content>empty</body-content>

</tag>

3.在jsp页面的调用

<ct:ipTag/>

备注：注册标签继承类的重写主方法是重点，注册方法和注册函数类似，但有区别

## 标签体

1.创建class

一样的继承SimpleTagSupport 类，然后重写dotag方法

//创建一个输出流，该流中的内容可通过ToString()的方法获取到其文本形式

StringWriter sw=**new** StringWriter();

//获取标签体对象

JspFragment jf =**this**.getJspBody();

//将标签体内容写入到输出流

jf.invoke(sw);

//以文本的形式获取到输出流中的内容

String content=sw.toString();

//小写转大写

content= content.toUpperCase();

//将转换了的大写内容写入到标准输出流

**this**.getJspContext().getOut().print(content);

2.将里边的方法注册

<!-- 标签体 -->

<tag>

<description>描述可删</description>

<name>LowerToUpper</name>

<tag-class>com.lonhwin.tags.tagBody\_LowerToUpper</tag-class>

<!--

empty:表示当前标签没有标签体

scriptless：表示当前标签具有标签体。但该标签体中不能出现java脚本（java代码块、jsp表达式）

但可以出现EL表达式

JSP（已过时，不能使用）：表示当前标签具有标签体，将标签体内容原样显示到浏览器

tagdependent：表示当前标签具有标签体，将标签体内容原样显示到浏览器，即使标签体中写的是EL，也不会对EL进行计算

-->

<body-content>scriptless</body-content>

</tag>

3.在jsp页面的调用

<%

String a="sheep";

pageContext.setAttribute("a", a);

%>

<ct:LowerToUpper>${a }</ct:LowerToUpper>

## 带属性

1.创建class

//标签的属性，反映到标签处理器中，就是一个set属性

**private** **boolean** iof;

**public** **void** setIof(**boolean** iof) {

**this**.iof = iof;

}

@Override

**public** **void** doTag() **throws** JspException, IOException {

**if**(iof){

/\* 把标签里的内容输出出来

//获取标签体

JspFragment jf=this.getJspBody();

//将标签体写入到标准输出流

jf.invoke(this.getJspContext().getOut());

\*/

//以下代码等价于上边的代码

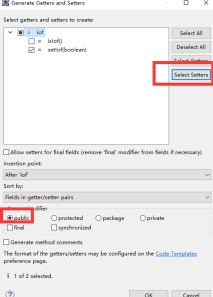
**this**.getJspBody().invoke(**null**);

}

}

备注：同样的，重写totag方法 要增属性的话，需要定义一个私有方法，并set他

进get set 框 select setters



2.将里边的方法注册

<!-- 带属性的标签 -->

<tag>

<description>描述可删</description>

<name>if</name>

<tag-class>com.lonhwin.tags.attribute\_if</tag-class>

<body-content>scriptless</body-content><!-- 解析EL表达式 -->

<attribute>

<name>iof</name>

<required>true</required><!-- 是否是必填的 -->

<rtexprvalue>true</rtexprvalue>

<!-- runtime expression value 若为true，则该属性的值支持EL与JSP表达式-->

</attribute>

</tag>

备注：attribute是关键，里边的每一个属性也都得注册

3.在jsp页面的调用

<%

**boolean** iof=**true**;

pageContext.setAttribute("iof", iof);

%>

<ct:if iof=*"*${iof}*"*>男魂牵梦萦塔顶塔顶霜期${iof}</ct:if><br/>

<ct:if iof=*"*${**not** iof }*"*>男魂牵梦萦塔顶塔顶霜期</ct:if><br/>

<ct:if iof=*"*<%=iof %>*"*>男魂牵梦萦塔顶塔顶霜期</ct:if><br/>

## List的forEachList

1.创建class

**private** List items;

**private** String var; //当前遍历对象要放入域中的属性名

**public** **void** setItems(List items) {

**this**.items = items;

}

**public** **void** setVar(String var) {

**this**.var = var;

}

@Override

**public** **void** doTag() **throws** JspException, IOException {

JspWriter out= **this**.getJspContext().getOut();

**for**(Object item: items)

{

**this**.getJspContext().setAttribute(var, item);

//<ct:forEachList items="${lista}" var="lis">

//${lis }

//前台 var值是x，后边用${x }调取

//之所以可以这样调取，是因为这句把var的具体前台赋值的名称定义到了四大域的page域里

**this**.getJspBody().invoke(**null**);

}

}

备注：**this**.getJspContext().setAttribute(var, item); 这句话用很简单，要仔细理解

1. 将里边的方法注册

<!-- List的forEachList -->

<tag>

<description>描述可删</description>

<name>forEachList</name>

<tag-class>com.lonhwin.tags.ForEachListTag</tag-class>

<body-content>scriptless</body-content><!-- 解析EL表达式 -->

<attribute>

<name>items</name>

<required>true</required><!-- 是否是必填的 -->

<rtexprvalue>true</rtexprvalue><!-- 支持EL与JSP表达式-->

</attribute>

<attribute>

<name>var</name>

<required>true</required><!-- 是否是必填的 -->

<rtexprvalue>false</rtexprvalue><!-- 不支持EL与JSP表达式-->

</attribute>

</tag>

3.在jsp页面的调用

<%

List<String> lista=**new** ArrayList();

lista.add("张三");

lista.add("李四");

lista.add("王五");

pageContext.setAttribute("lista",lista);

%>

<ct:forEachList items=*"*${lista}*"* var=*"lis"*>

${lis }<br>

</ct:forEachList>

## Set和Map的forEach