# 1.简介

EL表达式主要是用来替代jsp表达式,因为开发jsp页面的时候

遵守原则:尽量少些甚至不写java代码

# 2.语法

## 2.1基本写法

EL表达式作用:向浏览器输出域对象中的变量或者表达式计算的结果

**基本语法:${变量或者表达式} 代替<%=变量或表达式%>**

1)从四个域中自动搜索

|  |
| --- |
| <%  pageContext.setAttribute("name","小明");  %>  ${name}  <%--等价于下面的--%>  <%=pageContext.findAttribute("name")%> |

2)从指定域中取出数据

pageScope:page域

requestScope:request域

sessionScope:session域

applicationScope:application域

|  |
| --- |
| <%  pageContext.setAttribute("name","小明",PageContext.REQUEST\_SCOPE);  %>  ${requestScope.name} |

注意:

1)对于对象来说,对象.属性表示的是调用getXXX方法

|  |
| --- |
| <%  Student s = new Student();  s.setAge(200);  s.setName("lw");  pageContext.setAttribute("student",s,PageContext.REQUEST\_SCOPE);  %>  ${requestScope.student.name} //调用的是getName()方法 |

2)empty关键字可以判断对象是否为空或者字符串是否为””

|  |
| --- |
| <%  Integer a = null;  String b = null;  String c = "";  %>  ${empty a} <br>  ${empty b} <br>  ${empty c} <br> |

# 3.EL内置对象(11个)

pageContext

pageScope

requestScope

sessionScope

applicationScope

param :jsp 页面提交到另一个jsp界面,另一个jsp页面能获取前一个jsp页面的<input>标签的值,param是一个包含<input>标签的key value的map集合

paramValues:Map集合,value是一个数组

header :是一个Map,value是一个元素

headerValues:是一个Map,只不是他的value是一个数组

cookie:是一个Map集合,value是一个元素

initParam:是一个Map,value是一个元素

范例:

|  |
| --- |
| <%  String a = "hello,world";  pageContext.setAttribute("a",a,pageContext.REQUEST\_SCOPE);  %>  <%--获取上下文路径--%>  ${pageContext.request.contextPath}<br>  <%--根据不同的域获取数据--%>  ${requestScope.a}<br>  <%--参数取出--%>  <%=request.getParameter("name")%><br>  <%=request.getParameterValues("age")[0]%><br>  ${param['name']}<br>  ${param['age']}<br>  ${paramValues['age'][0]}  <%--获取请求头--%>  <%=request.getHeader("host")%><br>  <%=request.getHeaders("host").nextElement()%> <br>  ${header['host']}<br>  ${headerValues['host'][0]}<br>  <%--cookie--%>  <%=request.getCookies()[0].getName()%> - <%=request.getCookies()[0].getValue()%><br>  ${cookie["JSESSIONID"].name} - ${cookie['JSESSIONID'].value}<br>  <%--initparm--%>  <%=application.getInitParameter("a")%><br>  ${initParam['a']} |

# 4.js标签

jsp标签替换jsp脚本

## 4.1EL表达式的局限

不能条件判断,不能赋值,不能迭代,而el标签能做

## 4.2jsp标签分类

1)jsp内置标签(动作标签):使用的时候不需要导入标签库

2)jstl标签(标准标签):使用的时候都需要导入标签库

3)自定义标签: 使用的时候都需要导入标签库

### 4.2.1动作标签

<jsp:forward/>转发标签

<jsp:param/>参数标签

<jsp:include/>包含标签

|  |
| --- |
| <%--<jsp:forward/>转发--%>  <%  // request.getRequestDispatcher("/03.jsp").forward(request,response);  %>  <%--  <jsp:forward page="05action.jsp">  <jsp:param name="name" value="eric"></jsp:param>  <jsp:param name="name" value="jacky"></jsp:param>  </jsp:forward>  --%> |

|  |
| --- |
| <jsp:include page="/common/head.jsp"></jsp:include> |

注意:包含标签的原理是包含与被包含的页面单独翻译成不同的java文件,再运行时合并在一起

静态包含 vs 动态包含

静态包含:先合并在翻译,不能携带参数

动态包含:先翻译再合并,携带参数

### 4.2.2JSTL标签介绍

全称java standard tag library java标准库标签

特点:在使用jstl标签时需要导入标签库

jstl标签库分类:

1. 核心标签库(重要)(c)
2. 国际化标签库(fmt),还有格式化的功能,这个内容在”国际化.doc”里有
3. EL函数库(fn)
4. SQL标签库(sql)
5. XML标签库(x)

### 4.2.3JSTL标签使用

需要在jsp的顶部使用taglib指令导入需要的标签库

到 <http://archive.apache.org/dist/jakarta/taglibs/standard/binaries/>下载(jakarta-taglibs-standard-current.zip)

解压后在jakarta-taglibs-standard-1.1.2/lib/下的两个jar文件：standard.jar和jstl.jar文件拷贝到/WEB-INF/lib/下

1）核心标签库

|  |
| --- |
| <%@taglib prefix="c" uri="http://java.sun.com/jstl/core\_rt"%>  保存数据  <c:set></c:set>  获取数据  <c:out value=""></c:out>  单条件判断  <c:if test=""></c:if>  多条件判断  <c:choose></c:choose>  <c:when test=””></c:when>  <c:otherwise></c:otherwise>  用于迭代  <c:forEach></c:forEach>  <c:forTokens items="" delims=""></c:forTokens>  重定向  <c:redirect url=""></c:redirect> |

### 4.2.4<c:set>(核心)

|  |
| --- |
| <c:set var="name" value="eric" scope="page"></c:set>  <%--  var:数据名称  value:保存的数据值  scope:保存到哪个域page(默认),request,session,application  --%>  <%--//和上面的等价--%>  <%--// String name = "eric";--%>  <%--// pageContext.setAttribute("name",name);--%>  ${pageScope.name} |

### 4.2.5<c:out>(核心)

|  |
| --- |
| <%--<c:set var="name" value="" scope="page"></c:set>--%>  <c:out value="${name}" default="hahah"></c:out>  <c:out value="${name}" default="<h3>标题3</h3>" escapeXml="false"></c:out>  ${name}  <%--  value:代表获取域中的某个名称内容  default:默认值,如果value的内容为null,则使用默认值代替  escapeXml:默认为true,会进行转义,如果不要转义,这个值为false  --%> |

### 4.2.6<c:if>(核心)单条件

|  |
| --- |
| <c:if test="${10>8}">  <input type="submit" value="提交">  </c:if>  <%--  test:条件表达式,返回true就会执行if标签体的内容  --%> |

### 4.2.7choose when otherwise多条件

|  |
| --- |
| <%--  模拟登陆  登陆之后,一定会把数据存放到session域,user名称  --%>  <c:set var="user" value="eric" scope="session"></c:set>  <c:choose>  <c:when test="${!empty sessionScope.user}">  欢迎回来,你也用户名是:eric,<a href="">[退出登陆]</a>  </c:when>  <c:otherwise>  请<a href="">注册</a>或<a href="">登陆</a>  </c:otherwise>  </c:choose> |

when otherwise 相当于if else

### 4.2.8<c:forEach>(核心)

|  |
| --- |
| <%  List<Student> list = new ArrayList<>();  list.add(new Student("eric",12));  list.add(new Student("lily",13));  list.add(new Student("tom",14));  list.add(new Student("hah",15));  pageContext.setAttribute("list",list,pageContext.PAGE\_SCOPE);  Map<String,Student> map = new HashMap<>();  map.put("001",new Student("eric",12));  map.put("002",new Student("lily",13));  map.put("003",new Student("tom",14));  map.put("004",new Student("hah",15));  pageContext.setAttribute("map",map,pageContext.PAGE\_SCOPE);  %>  List集合:<br>  <%--  begin:从哪个元素开始遍历,从0开始  end:到哪个元素为止(不包括end下标对应的元素)  step:增加的步长  items:需要遍历的数据(数组|集合)  varStatus:当前状态对象,该对象封装当前元素状态信息,例如count属性:表示哪个元素,从1开始  --%>  <c:forEach begin="0" end="${list.size()}" step="1" items="${pageScope.list}" var="student" varStatus="varSta">  序号:${varSta.count}-姓名:${student.name} - 年龄: ${student.age} <br>  </c:forEach>  <c:forEach begin="0" end="${map.size()}" step="1" items="${pageScope.map}" var="entry" varStatus="varSta">  序号:${varSta.count} -编号:${entry.key}-姓名:${entry.value.name} - 年龄: ${entry.value.age} <br>  </c:forEach> |

### 4.2.9<c:f forTokens>

用于字符串切割

|  |
| --- |
| <%--  items:需要遍历的字符串  delims:指定分隔符  var:每个内容的名称  --%>  <c:forTokens items="${str}" delims="-" var="s">  ${s},  </c:forTokens> |

### 4.2.10<c:rediect>重定向

|  |
| --- |
| <%  // response.sendRedirect(request.getContextPath()+"/url.html");  %>  <%--替换上面的代码--%>  <c:redirect url="/url.html"></c:redirect> |

### 4.2.11自定义标签

1)首先开发标签处理程序,是一个普通java类,继承SimpleTagSupport类,覆盖doTag方法

2)在项目的WEB-INF目录下建立一个tld文件,例如lw.tld,内容参考核心标签库c.tld

3)在jsp页面导入自定义的标签库<%@taglib prefix="lw" uri="http://localhost"%>

4)使用标签<lw:showIp></lw:showIp>

步骤1): 自定义标签处理程序

|  |
| --- |
| /\*\*  \* 自定义标签处理程序  \*/  public class ShowIpTag extends SimpleTagSupport{  //下面的这个定义私有变量和公开的函数在SimpleTagSupport里有,一模一样  // private JspContext jspContext;  //用于设置PageContext对象  //JspContext是PageContext的父类,传进去的pc是,PageContext  // @Override  // public void setJspContext(JspContext pc) {  // this.jspContext = pc;  // }  @Override  public void doTag() throws JspException, IOException {  // PageContext pageContext = (PageContext)this.jspContext;  PageContext pageContext = (PageContext)this.getJspContext();  //通过PageContext获取到其他8个内置对象  HttpServletRequest request = (HttpServletRequest)pageContext.getRequest();  String ip = request.getRemoteHost();  //输出给浏览器  JspWriter out = pageContext.getOut();  out.write("当前客户端ip为:" + ip);  }  } |

范例:2)lw.tld文件

|  |
| --- |
| <?xml version="1.0" encoding="ISO-8859-1" ?>  <!DOCTYPE taglib  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"  "http://java.sun.com/dtd/web-jsptaglibrary\_1\_2.dtd">  <taglib>  <!--标签的版本号-->  <tlib-version>1.1</tlib-version>  <!--jsp版本号-->  <jsp-version>1.2</jsp-version>  <!--prefix-->  <short-name>lw</short-name>  <!--uri-->  <uri>http://localhost</uri>  <!--定义一个标签-->  <tag>  <!--标签名-->  <name>showIp</name>  <!--标签处理程序-->  <tag-class>com.company.tag.ShowIpTag</tag-class>  <!--标签体内容输出格式-->  <body-content>scriptless</body-content>  </tag>  </taglib> |

步骤3):导入和使用

|  |
| --- |
| <%@ page contentType="text/html;charset=UTF-8" language="java"%>  <%@taglib prefix="c" uri="http://java.sun.com/jstl/core\_rt" %>  <%@taglib prefix="lw" uri="http://localhost"%>  <html>  <head>  <title>核心标签</title>  </head>  <body>  <%--  showIP标签作用:用来显示客户端的ip地址  --%>  <lw:showIp></lw:showIp>  <%--替换上面的代码--%>  </body>  </html> |

### 4.2.12自定义标签执行过程

访问:http://localhost:8080/web/core.jsp

1. tomcat启动的时候,读取WEB-INF文件夹,包括web.xml和lw.tld文件
2. 翻译java文件,编译为class文件->构造方法,调用jspService()方法
3. 在内存tld文件中,查询是否存在uri为<http://localhost>名字的tld文件
4. 然后得到uri对应的lw.tld文件的内容
5. 当访问core.jsp时,读取到<lw:showIp>标签时,会根据lw->本页的uri->到对应的tld文件->tag标签->name标签里是否有内容为showIp->找到tag-class标签里的com.company.tag.ShowIpTag字符串

### 4.2.13自定义标签生命周期

Class SimpleTagSuppoert

实现了SimpleTag和JspTag接口

实际上就是研究SimpleTag的生命周期

由tomcat服务器调用

1. 构造方法:构造标签处理对象
2. void setJspContext(JspContext pc) 传入PageContext对象,JspContext是PageContext的父类,可以使用getJspContext()方法获取PageContext对象
3. void setParent(JspTag parent) 传入父标签对象,如果没有父标签,则不调用此方法,使用

getParent()方法获取父标签

1. void setJspBody(JspFragment jspBody) 传入标签体内容,如果没有标签体,则不调用此方法,可以用getJspBody()来获取标签体内容
2. void doTag()调用标签时执行的方法,我们的业务逻辑就写在这个方法中

### 4.2.14自定义标签的作用

1)控制标签体的内容是否输出

2)控制标签后面的内容是否输出

3)重复控制标签体内容的输出

4)改变标签体内容的输出

1)是否输出

|  |
| --- |
| /\*\*  \* 1.控制标签体内容是否输出  \* 输出:jspBody.invoke(null);  \* 不输出:不调用invoke方法  \*/  @Override  public void doTag() throws JspException, IOException {  //1.得到标签体内容  JspFragment jspBody = this.getJspBody();  //2.输出标签体内容  //void invoke(Writer writer);执行此方法,  //可以把标签体内容往给定的Writer输出  //得到JspWriter  //向浏览器输出标签体内容  // jspBody.invoke(jspBody.getJspContext().getOut());  jspBody.invoke(null);//等价于上面的,往浏览器输出  } |

2)控制标签后面的内容是否输出

|  |
| --- |
| /\*\*  \* 1.控制标签体后面的内容  \* 输出:什么都不做  \* 不输出:抛出异常SkipPageException  \*/  @Override  public void doTag() throws JspException, IOException {  throw new SkipPageException();  } |

3)重复输出标签体内容

|  |
| --- |
| /\*\*  \* 3)重复输出标签提内容  \*/  @Override  public void doTag() throws JspException, IOException {  JspFragment jspBody = this.getJspBody();  for(int i=0;i<5;i++){  jspBody.invoke(null);  }  } |

4)修改标签体内容再输出

|  |
| --- |
| /\*\*  \* 3)改变标签体内容  \*/  @Override  public void doTag() throws JspException, IOException {  //1得到标签体内容  JspFragment jspBody = this.getJspBody();  //1.1创建一个StringWriter,做一个缓冲区  StringWriter sw = new StringWriter();  //1.2把标签体的内容拷贝到StringWriter中  jspBody.invoke(sw);  //1.3得到StringWriter的内容  String content = sw.toString(); //返回缓冲区的内容  //2.修改标签体内容  content = content + "hello,world";  //3.输出修改后的内容  this.getJspContext().getOut().write(content);  } |

### 4.2.15自定义带属性的标签

1)在标签中,添加一个setXXX方法,用于给属性赋值,当然还要添加一个私有属性

public void setNum(int num){

this.num = num;

}

2)在tld文件中所在的表标签中配置

|  |
| --- |
| <?xml version="1.0" encoding="ISO-8859-1" ?>  <!DOCTYPE taglib  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"  "http://java.sun.com/dtd/web-jsptaglibrary\_1\_2.dtd">  <taglib>  <!--标签的版本号-->  <tlib-version>1.1</tlib-version>  <!--jsp版本号-->  <jsp-version>1.2</jsp-version>  <!--prefix-->  <short-name>lw</short-name>  <!--uri-->  <uri>http://localhost</uri>  <!--定义一个标签-->  <tag>  <!--标签名-->  <name>hello</name>  <!--标签处理程序-->  <tag-class>com.company.tag.HelloTag</tag-class>  <!--标签体内容输出格式-->  <body-content>scriptless</body-content>  <!--配置属性-->  <attribute>  <!--属性名称-->  <name>num</name>  <!--属性是否必须的,true:一定要,false:可以不要-->  <required>true</required>  <!--该属性值是否可以写EL表达式的内容,true:可以写el,false:不能写el-->  <rtexprvalue>false</rtexprvalue>  </attribute>  </tag>  <tag>  <name>parent</name>  <tag-class>com.company.tag.ParentTag</tag-class>  <body-content>scriptless</body-content>  </tag>  </taglib> |

## 4.3标签体内容的输出格式

|  |
| --- |
| <!--标签体内容输出格式-->  <body-content>scriptless</body-content> |

1)标签体内容可以支持和输出jsp表达式内容(注意:这个配置只能在传统标签中使用)

2)scriptless:标签体内容不支持jsp表达式内容

|  |
| --- |
| <lw:hello num="5">aaaaa <%="hellow,wrold"%></lw:hello> 报错 |

3)empty:该标签体内容必须是空的

两种写法:

|  |
| --- |
| <lw:hello></lw:hello>  <lw:hello/> |

4)tagdependent:标签体内容支持jsp表达式,不输出jsp表达式内容,把jsp表达式当作字符串处理

# 5.实战

## 5.1choose仿写

jsp文件:

|  |
| --- |
| <%@ page contentType="text/html;charset=UTF-8" language="java"%>  <%@taglib prefix="c" uri="http://java.sun.com/jstl/core\_rt" %>  <%@taglib prefix="lw" uri="http://localhost"%>  <html>  <head>  </head>  <body>  <lw:choose>  <lw:when test="${10<8}">  条件成立  </lw:when>  <lw:otherwise>  条件不成立  </lw:otherwise>  </lw:choose>  </body>  </html> |

ChooseTag文件

|  |
| --- |
| package com.company.tag;  import javax.servlet.jsp.JspException;  import javax.servlet.jsp.tagext.SimpleTagSupport;  import java.io.IOException;  public class ChooseTag extends SimpleTagSupport {  private boolean flag;  public void setFlag(boolean flag) {  this.flag = flag;  }  public boolean isFlag() {  return flag;  }  @Override  public void doTag() throws JspException, IOException {  //输出标签体的内容即可  this.getJspBody().invoke(null);  }  } |

WhenTag文件

|  |
| --- |
| package com.company.tag;  import javax.servlet.jsp.JspException;  import javax.servlet.jsp.tagext.SimpleTagSupport;  import java.io.IOException;  public class WhenTag extends SimpleTagSupport{  private boolean test;  public void setTest(boolean test) {  this.test = test;  }  @Override  public void doTag() throws JspException, IOException {  //根据test的值来输出标签体的内容  if(test){  this.getJspBody().invoke(null);  }  //得到父标签choose  ChooseTag parent = (ChooseTag)this.getParent();  parent.setFlag(test);  }  } |

OtherWiseTag文件

|  |
| --- |
| package com.company.tag;  import javax.servlet.jsp.JspException;  import javax.servlet.jsp.tagext.SimpleTagSupport;  import java.io.IOException;  public class OtherwiseTag extends SimpleTagSupport{  @Override  public void doTag() throws JspException, IOException {  //得到父标签choose  ChooseTag parent = (ChooseTag)this.getParent();  boolean test = parent.isFlag();  if(!test){  this.getJspBody().invoke(null);  }  }  } |

lw-tld

|  |
| --- |
| <?xml version="1.0" encoding="ISO-8859-1" ?>  <!DOCTYPE taglib  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"  "http://java.sun.com/dtd/web-jsptaglibrary\_1\_2.dtd">  <taglib>  <!--标签的版本号-->  <tlib-version>1.1</tlib-version>  <!--jsp版本号-->  <jsp-version>1.2</jsp-version>  <!--prefix-->  <short-name>lw</short-name>  <!--uri-->  <uri>http://localhost</uri>  <tag>  <name>choose</name>  <tag-class>com.company.tag.ChooseTag</tag-class>  <body-content>scriptless</body-content>  </tag>  <tag>  <name>when</name>  <tag-class>com.company.tag.WhenTag</tag-class>  <body-content>scriptless</body-content>  <attribute>  <name>test</name>  <required>true</required>  <rtexprvalue>true</rtexprvalue>  </attribute>  </tag>  <tag>  <name>otherwise</name>  <tag-class>com.company.tag.OtherwiseTag</tag-class>  <body-content>scriptless</body-content>  </tag>  </taglib> |

## 5.2forEach仿写

jsp文件:

|  |
| --- |
| <%@ page import="com.company.Student" %>  <%@ page import="java.util.List" %>  <%@ page import="java.util.ArrayList" %>  <%@ page import="java.util.Map" %>  <%@ page import="java.util.HashMap" %>  <%@ page contentType="text/html;charset=UTF-8" language="java"%>  <%@taglib prefix="c" uri="http://java.sun.com/jstl/core\_rt" %>  <%@taglib prefix="lw" uri="http://localhost"%>  <html>  <head>  </head>  <body>  <%  List<Student> list = new ArrayList<>();  list.add(new Student("ecic",1));  list.add(new Student("jscky",2));  list.add(new Student("rose",3));  list.add(new Student("hahah",4));  pageContext.setAttribute("list",list);  Map<String,Student> map = new HashMap<>();  map.put("001",new Student("lily",11));  map.put("002",new Student("tom",12));  map.put("003",new Student("make",14));  pageContext.setAttribute("map",map);  %>  <lw:forEach items="${list}" var="student">  姓名:${student.name} 年龄:${student.age}<br>  </lw:forEach>  <hr/>  <lw:forEach items="${map}" var="entry">  编号:${entry.key} - 姓名:${entry.value.name} - 年龄${entry.value.age}<br>  </lw:forEach>  </body>  </html> |

ForEachTag文件

|  |
| --- |
| public class ForEachTag extends SimpleTagSupport{  private Object items;//List或Map,需要变量的对象  private String var;//每个元素的名称  public void setVar(String var) {  this.var = var;  }  public void setItems(Object items) {  this.items = items;  }  @Override  public void doTag() throws JspException, IOException {  JspContext jspContext = this.getJspContext();  PageContext pageContext = (PageContext)jspContext;  //如果是List  if(this.items instanceof List){  List list = (List)items;  //遍历List  for(Object obj:list){  //把obj放入到域对象中  pageContext.setAttribute(this.var,obj);  //把标签体内容输出到浏览器  this.getJspBody().invoke(null);  }  }  //如果是Map  if(this.items instanceof Map){  Map map = (Map)items;  //遍历  Set set = map.entrySet();  for(Object obj:set){  //把obj放入到域对象中  pageContext.setAttribute(this.var,obj);  //把标签体内容输出到浏览器  this.getJspBody().invoke(null);  }  }  }  } |

lw.tld文件:

|  |
| --- |
| <tag>  <name>forEach</name>  <tag-class>com.company.tag.ForEachTag</tag-class>  <body-content>scriptless</body-content>  <attribute>  <name>var</name>  <required>true</required>  <rtexprvalue>false</rtexprvalue>  </attribute>  <attribute>  <name>items</name>  <required>true</required>  <rtexprvalue>true</rtexprvalue>  </attribute>  </tag> |

## 5.3用户登陆标签

LoginTag文件:

|  |
| --- |
| package com.company.tag;  import javax.servlet.jsp.JspException;  import javax.servlet.jsp.tagext.SimpleTagSupport;  import java.io.IOException;  //登陆页面的标签  public class LoginTag extends SimpleTagSupport {  private String name;  private String password;  public void setPassword(String password) {  this.password = password;  }  public void setName(String name) {  this.name = name;  }  @Override  public void doTag() throws JspException, IOException {  String html = "";  html += "<table align='center' border='1' width='300px'";  html += "<caption>用户登陆页面</caption>";  html += "<tr><th>用户名:</th><td><input type='text' name='"+this.name+"'</td></tr>";  html += "<tr><th>密码:</th><td><input type='password' name='"+this.password+"'</td></tr>";  html += "<tr><td colspan='2' aglign='center'><input type='submit' value='登陆'/><input type='reset'/></td></tr>";  html += "</table>";  this.getJspContext().getOut().write(html);  }  } |

lw.tld文件

|  |
| --- |
| <tag>  <name>login</name>  <tag-class>com.company.tag.LoginTag</tag-class>  <body-content>scriptless</body-content>  <attribute>  <name>name</name>  <required>true</required>  <rtexprvalue>false</rtexprvalue>  </attribute>  <attribute>  <name>password</name>  <required>true</required>  <rtexprvalue>false</rtexprvalue>  </attribute>  </tag> |

jsp文件:

|  |
| --- |
| <lw:login name="userName" password="password"></lw:login> |

## 严重问题:

|  |
| --- |
| <%@taglib prefix="c" uri="http://java.sun.com/jstl/core" %>  <c:set var="name" value="eric" scope="page"></c:set>  <c:out value="${name}"></c:out> |

如果这么写的话会报错,不知道是什么原因,解决方法:

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