# EL表达式&JSTL

【学习目标】理解、了解、应用、记忆

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. 【理解】能够阐述EL表达式的作用
2. 【应用】能够使用EL表达式从域中取出数据
3. 【应用】熟悉EL的内置对象
4. 【理解】能够阐述jstl的作用
5. 【应用】掌握JSTL下载与导入
6. 【应用】掌握JSTL核心库的常用标签

# EL表达式&EL的内置对象（重点：必须掌握）

EL表达式的学习路线：

1 EL表达式的由来

2 使用EL获取web四个容器（page、request、session、serveltContext）中的数据

3 使用EL解析复杂数据（List、Map、数组、自定义对象）

4 使用EL执行运算（算数运算、逻辑运算、比较运算）

5 使用EL的内置对象（快速操作常用的javaweb对象）

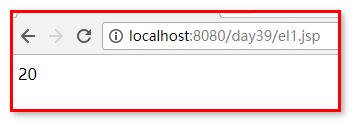
学习任何一个知识点都要从由来开始，EL也不例外，通过一个场景我们来体会一下：

场景：

在jsp设置四个数据（10 20 30 40 ）的向页面输出（10+20+（30-40））计算结果

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>Insert title here</title>  </head>  <body>  <%--  需求：在jsp设置四个数据（10 20 30 40 ）的向页面输出（10+20+（30-40））计算结果  问题：在页面输出数据的时候，操作非常麻烦，所以，需要一种新的技术，来解决这个问题——EL表达式  --%>  <%  pageContext.setAttribute("num1", 10);  pageContext.setAttribute("num2", 20);  pageContext.setAttribute("num3", 30);  pageContext.setAttribute("num4", 40);  %>  <%= (Integer)pageContext.getAttribute("num1") + (Integer)pageContext.getAttribute("num2") +((Integer)pageContext.getAttribute("num3") - (Integer)pageContext.getAttribute("num4")) %>  </body>  </html> |

效果：



需求：需要一种新的jsp页面使用java代码的方式，方便直接输出内容。

EL全称：Expression Language

作用：代替jsp中脚本表达式的功能，简化对java代码的操作。

具体简化的内容：

获取数据（获取四个容器中数据）

执行运算（EL执行运算）

快速操作常用的javaweb对象（11个内置对象）

EL表达式是什么：

EL表达式是jsp中的技术，书写格式为：${ 表达式}

表达式作用：要获取数据的名称、执行的运算、要获取的web常用对象。

pageContext:上下文？

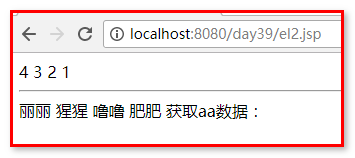
抽象概念：研究他的作用，使用作用代替文本概念：可以获取其他八个内置对象，可以给四个容器设置和获取数据，这个就是上下文对象。

## EL使用——获得四个容器中的数据（page\request\session\servletContext :域对象）的数据

代码演示：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>Insert title here</title>  </head>  <body>  <%--  需求：获取四个容器中的数据  --%>  <%  pageContext.setAttribute("name", "丽丽", 4);  pageContext.setAttribute("name", "猩猩", 3);  pageContext.setAttribute("name", "噜噜", 2);  pageContext.setAttribute("name", "肥肥", 1);    %>  <%=pageContext.APPLICATION\_SCOPE %>  <%=pageContext.SESSION\_SCOPE %>  <%=pageContext.REQUEST\_SCOPE %>  <%=pageContext.PAGE\_SCOPE %>    <hr>    <%-- ${applicationScope.name }  ${sessionScope.name }  ${requestScope.name }  ${pageScope.name } --%>  <%-- ${name } --%>  获取aa数据： ${aa }  </body>  </html> |

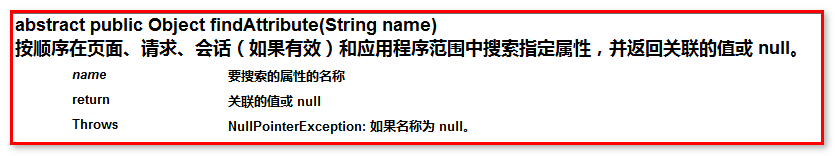
效果：



细节：使用pageContext的getAttribute方法或者findAttribute方法从4个容器中取出数据的时候，如果指定的key不存在会得到null，而使用el表达式取出的时候指定的key不存在，页面上什么都没有

其实还可直接这样写：

对应pageContext中的findAttribute（）方法

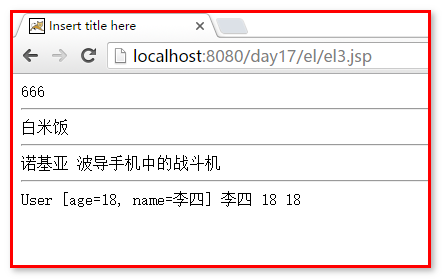


## EL使用——解析复杂数据

复杂数据：数组，集合（List Map），自定义对象（Person User animal）

|  |
| --- |
| <%-- <%@page import="cn.igeek.domain.User"%> --%>  <%-- page指令可以写多个 在import属性中，可以一次性导入多个包或则类--%>  <%@ page language=*"java"* import=*"java.util.\*,cn.igeek.domain.User"* pageEncoding=*"UTF-8"* %>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>Insert title here</title>  </head>  <body>  <%-- el获取复杂数据 --%>  <%  //演示数组  int[] arr = {666,1888,2333,1024,999};  pageContext.setAttribute("arr", arr);    %>  可以通过角标获取数据：${arr[0]}  <hr>  <%  //演示集合  List list = new ArrayList();  list.add("白米饭");  list.add("咸菜");  list.add("紫菜汤");  pageContext.setAttribute("list", list);    Map map = new HashMap();  map.put("sj1", "诺基亚");  map.put("sj2", "黑莓");  map.put("sj3", "摩托罗拉");  map.put("sj4", "大哥大");  map.put("aa.bb.cc", "波导手机中的战斗机");    pageContext.setAttribute("map", map);  %>  可以通过角标获取数据：${list[0] }  <hr>  通过map的key获取数据  ${map.sj1}  ${map["aa.bb.cc"]}  <hr>  <%-- EL获取自定义对象 --%>  <%  User u = new User();  u.setName("李四");  u.setAge(18);    pageContext.setAttribute("u", u);  %>  通过相同的属性名称获取数据：  ${u }  ${u.name }  ${u.age }  <%-- 所有可以使用点的地方，都可以使用【】获取数据 --%>  ${u["age"] }  </body>  </html> |

效果：



## EL使用——执行运算

1、支持算术运算符：

+ - \* / %

2、支持逻辑运算符：



3、支持比较运算符：



4、empty运算符：检查对象是否为null或“空”

对于自定义对象，检查是否为null

对于集合检查，是否为空（集合存在，但是没有数据）

代码：

|  |
| --- |
| <%  User user = new User();  user.setName("张三");  pageContext.setAttribute("user", user);  %>  ${empty user } |

效果：

显示：False

4、${user!=null?user.name : “”}三元运算

|  |
| --- |
| <%  User user = new User();  user.setName("张三");  pageContext.setAttribute("user", user);  %>  ${empty user }  <hr>  ${user!=null?user.name:"" } |

效果：

显示：张三

<%-- 演示EL执行运算 --%>

<%

int x = 3;

//String y = "4g";

int y = 4;

pageContext.setAttribute("x", x);

pageContext.setAttribute("y", y);

%>

<%-- 使用EL表达式获取数据，最终都是将数据转换成字符串，根据字符串，格式化数字，再进行计算 --%>

<%-- z的数据不存在，不存在的数据，不参与运算 --%>

${x+y }

${x-y }

${x\*y }

${x/y }

${x%y }

<hr>

<%

boolean flag = true;

boolean info = false;

pageContext.setAttribute("flag", flag);

pageContext.setAttribute("info", info);

%>

${flag & info}

<%-- ${ !info} --%>

${flag || info}

<hr>

${x < y }

${x <= y }

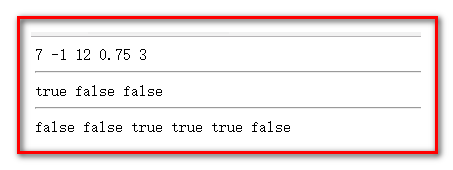
${x > y }

${x >= y }

${x == y }

${x != y }

效果：



注意：当算术运算的时候，如果字符串内容不能解析成数据，则报数字格式化异常。

数字格式化错误：EL取出来的数据无法转换成数字



## EL的11个内置对象使用

EL表达式它也有自己的内置(直接使用)对象可以直接在EL表达式中使用：

从不同的容器中取值

pageScope

requestScope

sessionScope

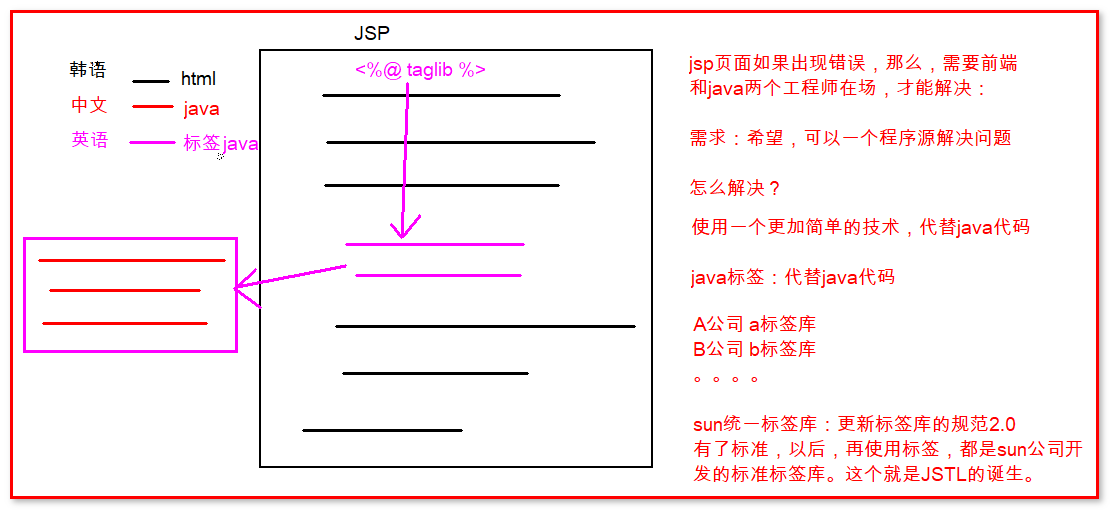
applicationScope

剩下7个介绍：

|  |  |
| --- | --- |
| param | 获取用户提交的请求参数  测试代码：  ${param }  效果： |
| paramValues | 获取某个key对应的多个value值（获取页面中checkbox）  测试代码：  ${paramValues }<%--相当于getParameterValues() --%>  ${paramValues.username[0] }  效果： |
| header | 获取请求头中的信息  测试代码：  ${header } |
| 效果： | |
| headerValues | 某个请求头中的多个value值  测试代码：  ${headerValues }<br><br>  ${headerValues.cookie[0] } |
|  | |
| cookie | 它获取到的一个cookie数组，获取所有的cookie数据  测试代码：  <%-- ${cookie }获取了cookie数组，如果获取数组中的指定cookie,使用点名称的方式获取 --%>  ${cookie }  ${cookie.JSESSIONID.name }  ${cookie.JSESSIONID.value } |
| Cookie测试效果： | |
| pageContext | 它就和JSP内置对象pageContext功能一致（获取其他内置对象：jsp）  测试代码：  <%-- 使用pageContext，获取了 request，调用request对象的方法--%>  ${pageContext.request.contextPath}  效果： |
| initParam | 获取的项目的全局配置参数  测试配置：  <!-- 设置项目的全局参数 -->  <context-param>  <param-name>sh</param-name>  <param-value>常州极客营</param-value>  </context-param>  <context-param>  <param-name>wh</param-name>  <param-value>武汉极客营</param-value>  </context-param>  测试代码： |

# JSTL的核心标签库使用（重点:必须掌握）

标签的由来回顾：



Jsp：html和java混杂，维护麻烦：

创建java标签，代替java代码，实现功能。

A公司：A标签库（库：一整套标签）

B公司：B标签库

C公司：C标签库

标签库版本太多，到底用那个?

一些优秀的标签，随着，公司的解体，没有人维护（不更新）

Java标签库，没有统一的管理！！！

Sun公司，推出了一套java标签的标准，根据这个标准，自己制作了一套标签，给所有的Java程序员使用，这个标签就是JSTL。

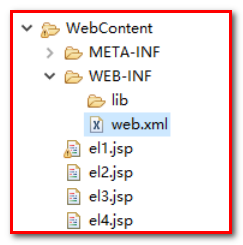
JSTL的介绍：



学习的资料：通过《第8章 标准标签库\_0519.doc》这篇文档进行学习

创建新工程：

1. 将jar包导入工程



2）API文档

资料--》第8章 标准标签库\_0519.doc

手动测试

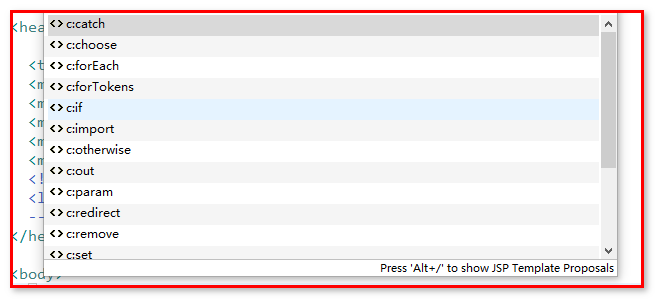
1. 使用核心标签库，引入标签库到jsp页面中



prefix 表示的是 标签的小名。

Uri 表示的是 引入那个标签库

当我们使用快捷键出现提示的时候，表示引入标签库成功。



注意版本：



JSTL标签库提供5大功能（了解）：

JSTL标签库提供5大功能（了解）：

core：jstl的核心标签库，使用最多（必须掌握）

fmt：格式化的标签（对页面显示数据，格式化，现在都交给前端去做）

functions：jstl中提供对字符串操作的函数库(建议，在数据显示在页面之前，在后台程序中，先格式化好字符串，然后直接显示，不再页面做处理，如果有前端，交给前端处理（javascript 解析json格式数据）)

sql：jstl提供的在jsp页面上书写sql，操作数据库，目前已经不再（不允许）使用（jsp开发模式二（MVC），这个开发模式不允许，在页面操作数据库）

xml：jstl操作xml文件的。目前已经不再使用（页面传递数据，页面解析数据，json格式字符串。）

## c:if标签（重点：必须掌握）

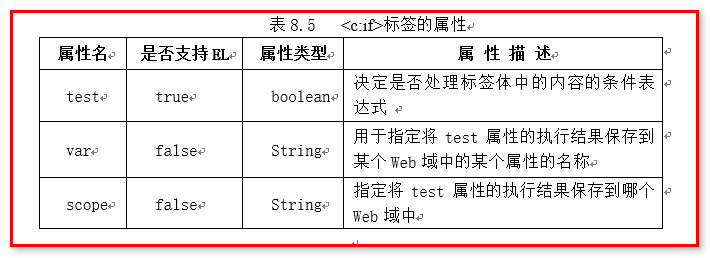
作用：相当于java代码中if语句

使用c:if 标签，在JSP页面上可以完成if判断。注意：在JSTL核心标签库中没有c:else.

只有if(){}结构 没有 if(){}else{}结构

写法：<c:if 属性=属性值。。。 > 条件成立的时候，执行标签中的内容 </c:if>

API截图：



人话翻译：

Test：属性:判断是否执行标签内的内容（true——执行标签中的内容，false，不执行）。

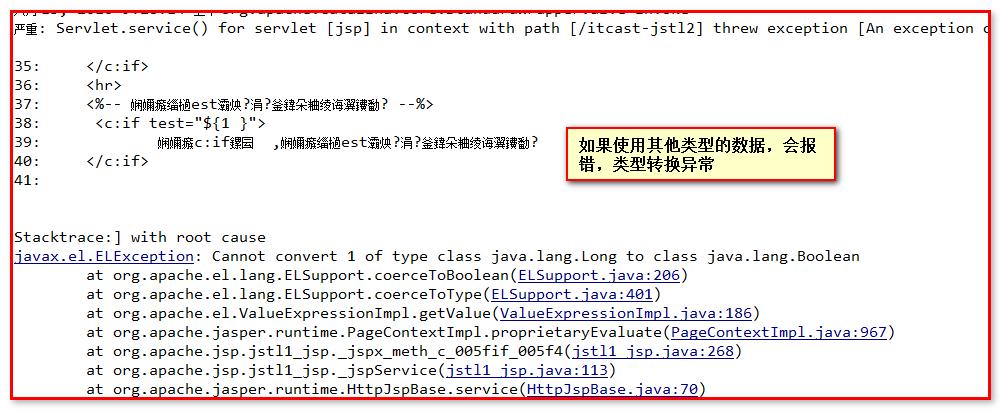
Var：用来保存test属性的结果（使用var属性给他取个名字），这个结果可以保存到指定的容器中。（如果没有指定容器，默认存入page容器中）

Scope：指定保存数据的容器。

是否支持EL表达式：当前这个属性中是否可以书写EL表达式

属性类型：当前属性可以接受一个什么类型的值

C:if标签异常，如果test属性接受一个不是boolean类型的数据，发生类型转换异常

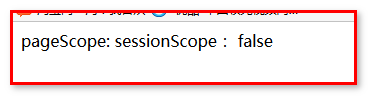


演示标签：

测试代码：

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>Insert title here</title>  </head>  <body>  <%-- var 设置一个变量空间，将test的结果保存下来，然后，默认存入page容器中 --%>  <%-- scope="session" 设置var中的数据保存在哪个容器中--%>  <c:if test=*"false"* var=*"hehe"* scope=*"session"*>  测试if标签，EL表达式方式  </c:if>  pageScope:${pageScope.hehe }  requestScope:${requestScope.hehe }  sessionScope:${sessionScope.hehe }  applicationScope:${applicationScope.hehe }  </body>  </html> |

效果：



## c:choose c:when c:otherwise（重点：必须掌握）

c:choose c:when c:otherwise 相当于：

if(){

}else if(){

} else if(){

}

。。。。。

else{

}

<c:choose>标签它必须与<c:when>和<c:otherwise>标签一起使用，它表示哪些when和otherwise是在一组的。

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。

c:when ,c:otherwise 属于同一级别标签。同时是c:choose的子标签

测试代码：

<%-- 演示choose when otherwise 标签 --%>

<%

int num = 4;

pageContext.setAttribute("num", num);

int flag = 1;

pageContext.setAttribute("flag", flag);

%>

<c:choose>

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

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

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

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

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

<c:when test=*"*${num==6 }*"*>星期六</c:when>

<c:when test=*"*${num==7 }*"*>星期日</c:when>

<c:when test=*"*${flag==1 }*"*>白天</c:when>

<c:when test=*"*${flag==2 }*"*>黑夜</c:when>

<c:otherwise>参数不合法</c:otherwise>

</c:choose>

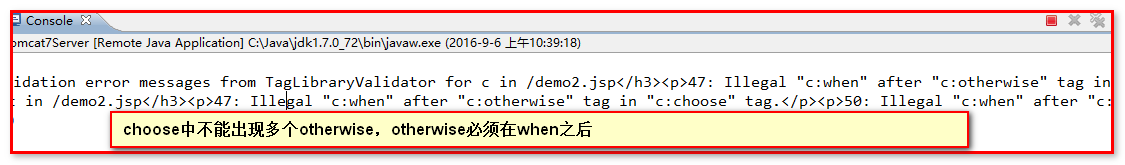
效果：



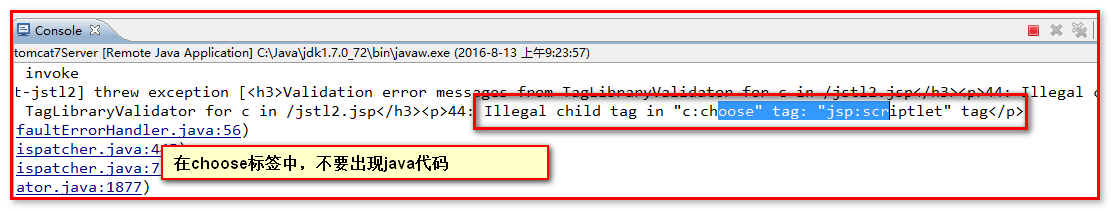
异常：使用when otherwise标签的时候，一定要写在choose标签内，不然报错



异常：一组choose中有多个otherwise，就出现异常。



异常：choose标签中不要写java代码

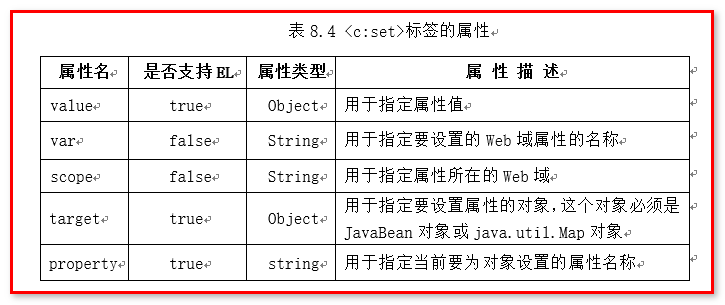


## c:set和c:out标签

c：set作用：它可以给某个容器中保存数据，或者去修改某个对象的属性值。

注：相当于代替了pageContext.setAttribute

API截图：



Var:声明了一个变量空间，用来存储数据(value属性的值)的

Value：要保存的数据

Scope：指定保存在那个容器中

Target：指定要修改的对象

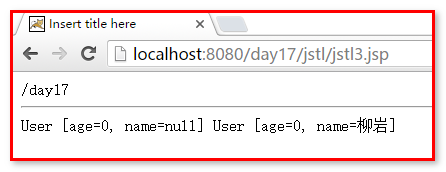
Property：指定要修改的属性

演示标签：

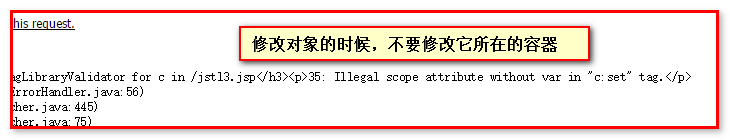
测试代码：

|  |
| --- |
| <%@page import=*"cn.igeek.domain.User"*%>  <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>Insert title here</title>  </head>  <body>  <%-- 演示set标签 --%>  <%--  int i = 0;  var="root" 声明一个变量空间，用来保存数据 （int i ）  value="${pageContext.request.contextPath }" 给var属性声明的变量空间赋值 ( = 0;)  --%>  <c:set var=*"root"* value=*"*${pageContext.request.contextPath }*"* scope=*"session"*></c:set>  ${sessionScope.root }  <hr>  <%-- 修改数据演示 --%>  <% pageContext.setAttribute("user", new User()); %>  <%--  target="${user }" 指定要修改的对象  property="name" 指定要修改的属性  --%>  ${user }  <c:set target=*"*${user }*"* property=*"name"* value=*"柳岩"*></c:set>  ${user }  </body>  </html> |

效果：



异常原因： 修改对象属性的时候不能修改它所在的容器。Var 属性和scope属性必须一起使用。

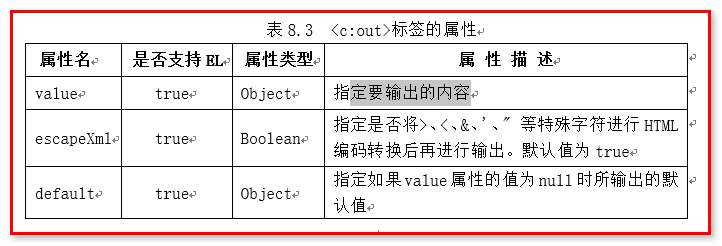


异常原因：Target属性和property属性是要一起使用的，如果修改的数据不是自定义对象，那么就会报错



c：out 作用：它可以把数据输出到页面上，相当于JSP的内置对象out

API截图：

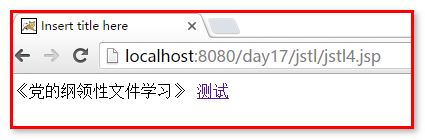


演示标签：

测试代码：

|  |
| --- |
| <%@page import=*"cn.igeek.domain.User"*%>  <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>Insert title here</title>  </head>  <body>  <%-- 演示out标签 --%>  <%-- default设置没有数据的时候，默认输出 --%>  <%  pageContext.setAttribute("str", null);  %>  <c:out value=*"*${str }*"* default=*"《党的纲领性文件学习》"*></c:out>  <%-- escapeXml="true" 让浏览器不解析其中的html--%>  <c:out value=*"*<a href=*'http://www.baidu.com'*>*测试*</a>*"* escapeXml=*"false"*></c:out>  </body>  </html> |

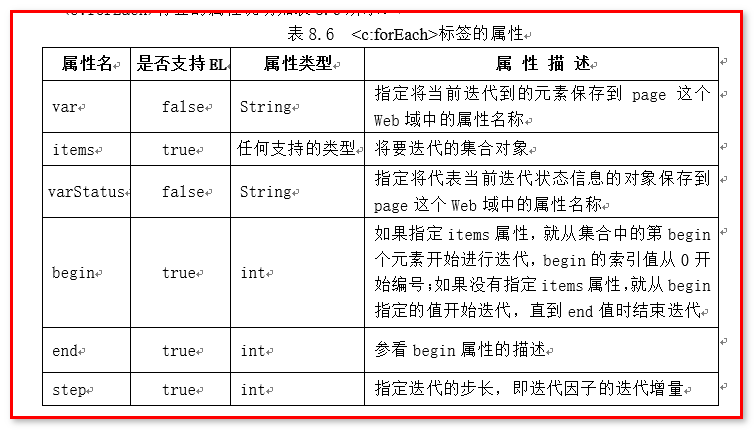
效果：



## c:forEach标签（重点：必须掌握）

c:forEach 循环的标签。 实现 java中for循环的功能

API截图：



varStatus :保存了当前循环过程中的信息

 1 public java.lang.Integer getBegin()

返回为标签设置的begin属性的值，如果没有设置begin属性则返回null

2 public int getCount()

返回当前已循环迭代的次数

3 public java.lang.Object getCurrent()

返回当前迭代到的元素对象

4 public java.lang.Integer getEnd()

返回为标签设置的end属性的值，如果没有设置end属性则返回null

5 public int getIndex()

返回当前迭代的索引号

6 public java.lang.Integer getStep()

返回为标签设置的step属性的值，如果没有设置step属性则返回null

7 public boolean isFirst()

返回当前是否是第一次迭代操作

8 public boolean isLast()

返回当前是否是最后一次迭代操作

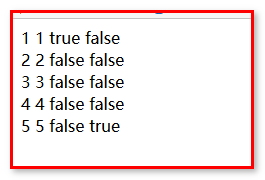
Foreach不循环对象（集合、数组）的情况下，单单控制循环的次数。

Var：用来设置，保存在page容器中的数据的名称

测试代码：

|  |
| --- |
| <%-- 演示foreach标签 --%>  <%  for (int i = 1; i <= 5; i++) {  }  %>  <%--  begin="1" int i = 1;  end="5" i<=5;  step="6" 步长，控制循环，间隔几次循环，执行一次循环体中的内容    step="1" 1 2 3 4 5 M M M M M  step="2" 12 34 5 M M M  step="3" 123 45 M M  step="4" 1234 5 M M  step="5" 12345 M    var 保存了控制循环变量数据的值（在foreach标签，不循环对象的时候）  varStatus 保存了循环过程中状态信息（索引号、循环总数、是否第一次或者最后一次）的对象  --%>  <c:forEach begin=*"10"* end=*"15"* step=*"1"* var=*"info"* varStatus=*"sta"*>  <td>${sta.index}</td>  <td>${sta.count}</td>  <td>${sta.first}</td>  <td>${sta.last}</td><br>  </c:forEach> |

效果：



Foreach循环对象的情况下（数组，集合：list和map）

测试代码：

<%-- 演示foreach，循环对象 --%>

<%

int[] arr = {666,23333,886,9527,82};

pageContext.setAttribute("arr", arr);

%>

<%-- var="num" :保存的是数组中的元素 --%>

<c:forEach items=*"*${arr }*"* var=*"num"*>

${num }

</c:forEach>

<hr>

<%

List list = new ArrayList();

list.add("跑男");

list.add("欢乐喜剧人");

list.add("极限挑战");

list.add("笑傲江湖");

list.add("爸爸去哪儿");

pageContext.setAttribute("list", list);

%>

<c:forEach items=*"*${list }*"* var=*"name"*>

${name }

</c:forEach>

<hr>

<%

Map map = new HashMap();

map.put("addr1", "东京");

map.put("addr2", "东莞");

map.put("addr3", "巴黎");

map.put("addr4", "东南亚");

pageContext.setAttribute("map", map);

%>

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

${entry.value }

${entry.key }

</c:forEach>

效果：



# 开发模式&JavaBean

## JavaBean介绍

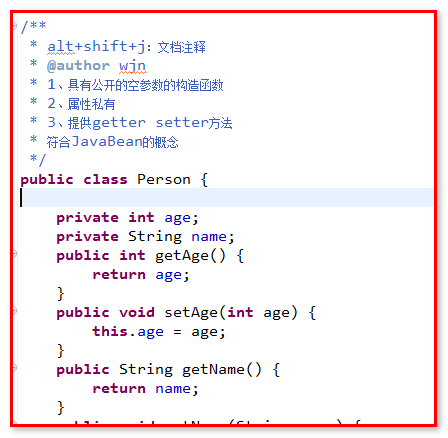
JavaBean：它是一个简单的Java类。属性要求是private 这个类中拥有get或set方法即可，但要求这个类必须有一个公开的空参数的构造函数。

特点：1、一定要有一个无参数的构造函数---public 无参的构造函数

2、属性必须是 private的。

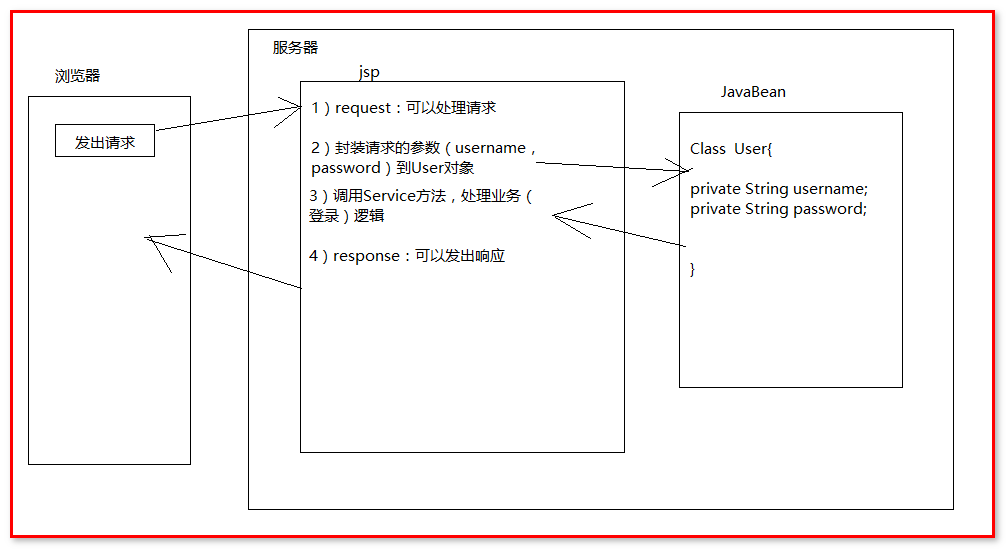
3、为这个私有的属性，提供公有的访问方法getter ,setter。

注意：目前使用的javaBean是用来封装数据的（处理业务逻辑，spring框架）。



提示：后期开发中，这些类一般会保存在名称为domain或者 beans.或者entity 或者pojo包中，这是一个开发的习惯，你在公司，拿到公司项目的所有封装数据的Javabean一般都会在这里包中。

## JSP开发模式一（早期开发模式，现在已经淘汰了）



在JSP刚刚推出的时候，基于JSP出现了JSP开发模式一：

使用JSP处理用户的请求以及响应，使用JavaBean来封装用户的数据

这种模式问题：

1. java代码、html代码、混杂在一起，代码阅读性差
2. 维护不方便（前端和后台开发人员都在的情况下，才能维护页面（jsp））
3. 耦合性高（处理请求，处理响应，处理业务方法 操作数据库方法都写在jsp中）

jsp开发模式一：它可以解决一些小型的需求，但是后期项目需求越来越复杂，这种开发模式已经不行了。JSP页面中的JAVA代码太多难以维护和升级。

耦合性高：所有的代码都拥挤到一个类或者jsp文件中，导致，只要某一个部分出问题，其他功能，也无法执行。

## MVC设计(开发)模式（jsp开发模式二）（重点：必须掌握）

掌握：可以画出MVC设计模式的流程图

MVC设计模式：

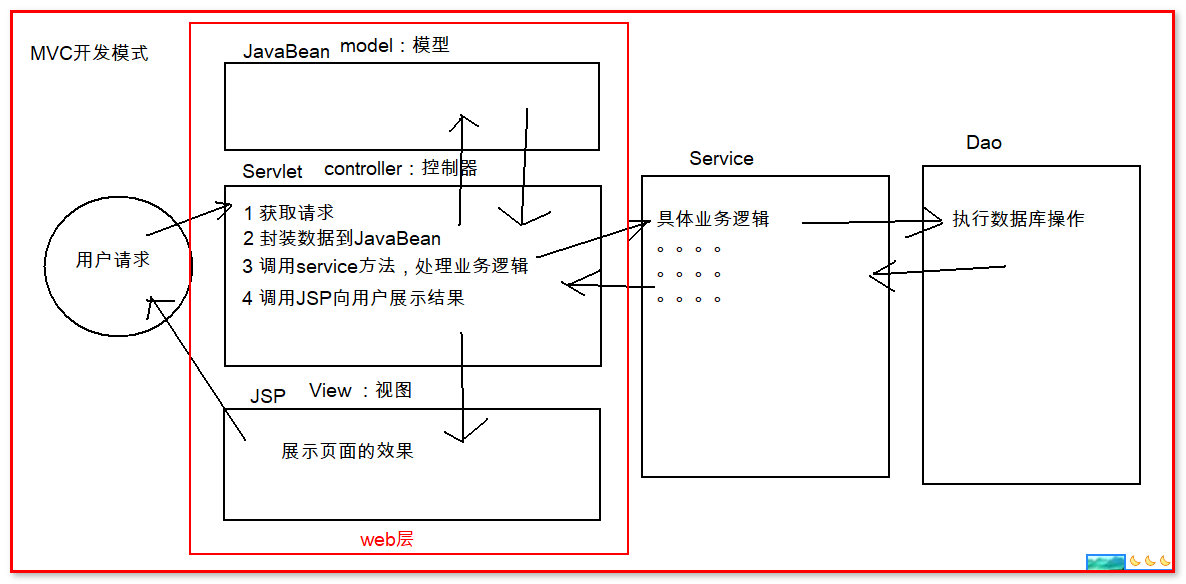
M：model 模型--- 数据模型（JAVABean）。作用： 存取数据、业务模型。

V：view 视图 主要是用来展示数据和页面效果的，jsp

C：Controller 控制器（servlet）。处理请求，调用业务处理方法，给出浏览器响应的作用

控制器：struts2：action springMVC：Controller

## JSP开发模式二：jsp model 2 MVC:



总结设计模式实现：

使用JSP显示数据，把JSP当做MVC中的view层。

使用JavaBean封装数据，它是MVC中的Model。模型层

使用Servlet处理用户的请求和响应。它是MVC中的Controller 控制层

优点：耦合性低，可维护性高。程序整体非常清晰。