# JSP详解

# 一、JSP

　　　　1、什么是JSP？

　　　　　　　JSP(Java Server Pages):是以Java语言为基础的动态网页开发技术，

　　　　　　　特点：

　　　　　　　　　　Servlet特点：在Java源码中嵌入html源码

　　　　　　　　　　JSP特点：在html源码中嵌入java代码

　　　　　　　JSP就是Servlet

　　　　　　　　　　1、tomcat获得JSP文件后，先将JSP转成servlet，变成xxx.java(servlet源码)，

                                    2、编译成.class文件后保存在work目录下。

                                    3、tomcat运行class文件然后输出浏览器。

二、JSP语法

 　　　　JSP模版数据：

　　　　　　　　就是JSP中的HTML代码，它的内容给是固定的，无论程序如何运行模版数据输出到客户端浏览器时都不会发生改变，当我们创建一个JSP时，模版就已经固定了。

　　　　 元素：JSP中的java部分，包括脚本(JavaScript，或者java代码)以及JSP指令(Directive)与JSP标签(Tag)等，元素决定着程序的流程，元素是不会显示到浏览器的。这几个都会在接下来讲解到

　　　　 JSP脚本：

　　　　　　　　1、使用<% 编写java代码 %>，中间java代码必须遵循Java语法，



　　　　　　　　　　　为什么能够使用out输出？这里就涉及到了JSP的九大内置对象了，后面会讲解到，到时候回过头来在看看这里，就会知道为什么可以使用。

　　　　　　　　　　　来看看，jsp变为servlet时的代码是如何编写的。



　　　　　　　　　　在JSP中JSP指令(后面会讲)：

　　　　　　　　　　　　　　　　<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

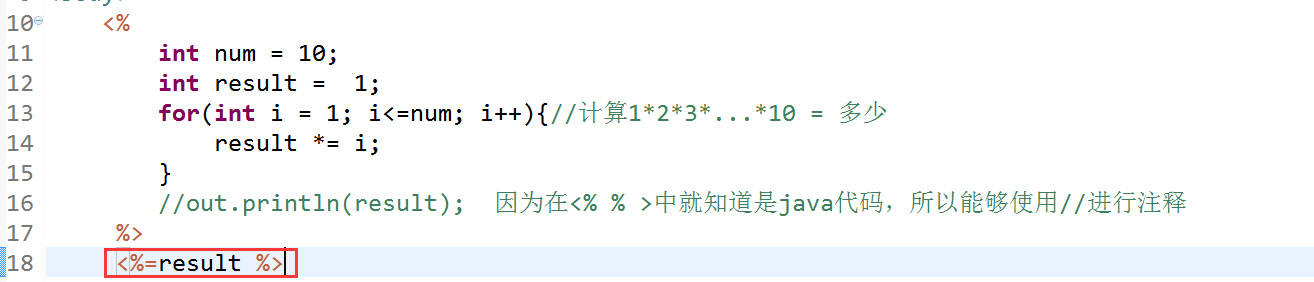
　　　　　　　　　　　　　　　　　　这句代码在servlet就变为了

　　　　　　　　　　　　　　　　response.setContentType("text/html; charset=UTF-8");//这句代码的意思就是通知tomcat和浏览器都使用UTF-8码表，并且发送回浏览器的数据类型是text/html。这是有JSP指令中画红色部分代码转变的，而指令中的pageEncoding="UTF-8"的意思是JSP转换为Servlet时采用UTF-8码表编码，因为可能JSP中包含中文。

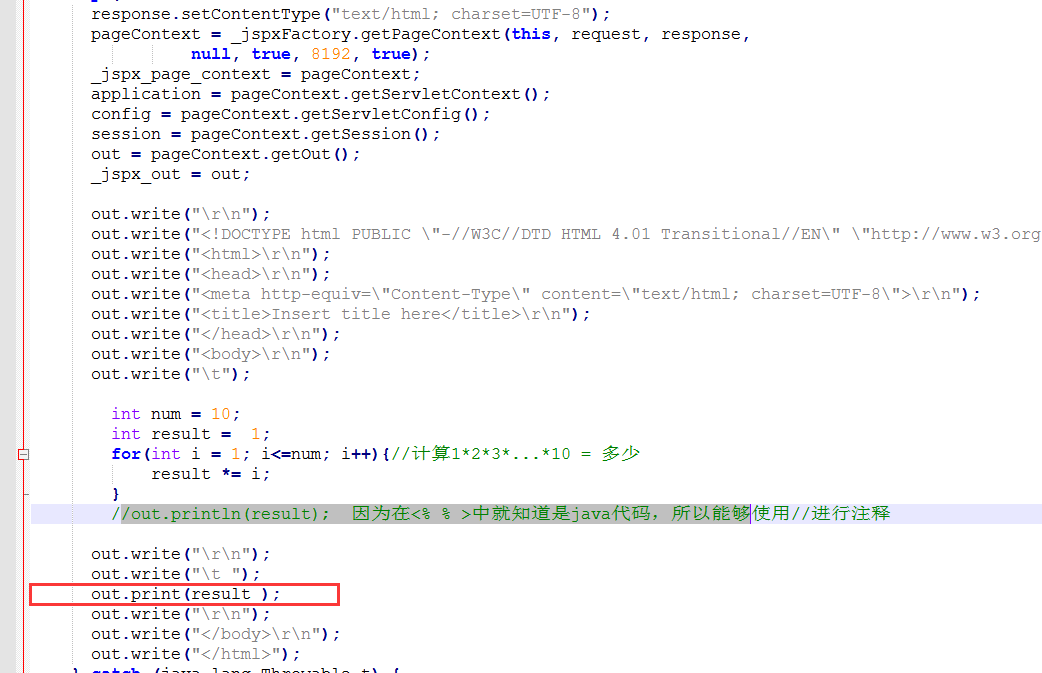
　　　　　　　　　　对于JSP模版数据来说，就原封不动的使用out.write()来输出到浏览器

　　　　　　　　　　而对于JSP元素来说，就会变成普通的java代码，因为在servlet中，就可以直接编写java代码。

2、使用<%=xxx %>来输出结果



　　　　　　　　　　使用<%=result %>来输出结果，servlet中就会将其转换为out.print(result)进行输出。输出各种类型数据：int、double、boolean、String、Object等



　3、JSP注释

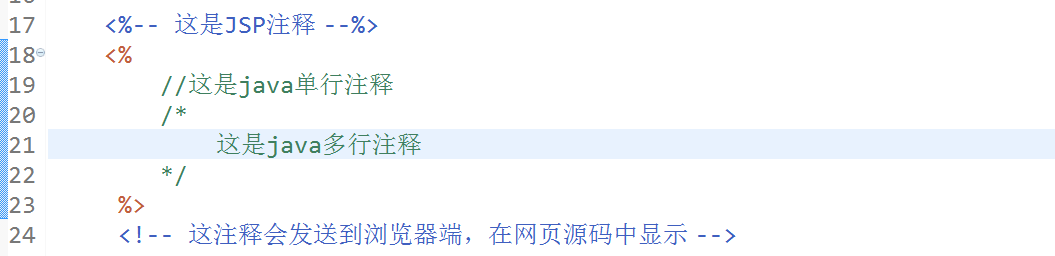
　　　　　　　　　　<%-- --%>　　：jsp注释，

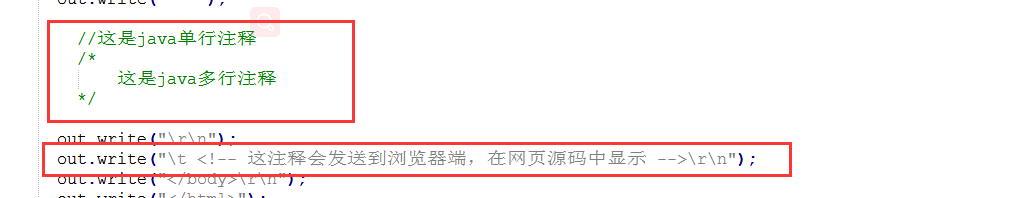
　　　　　　　　　　//　　：java单行注释

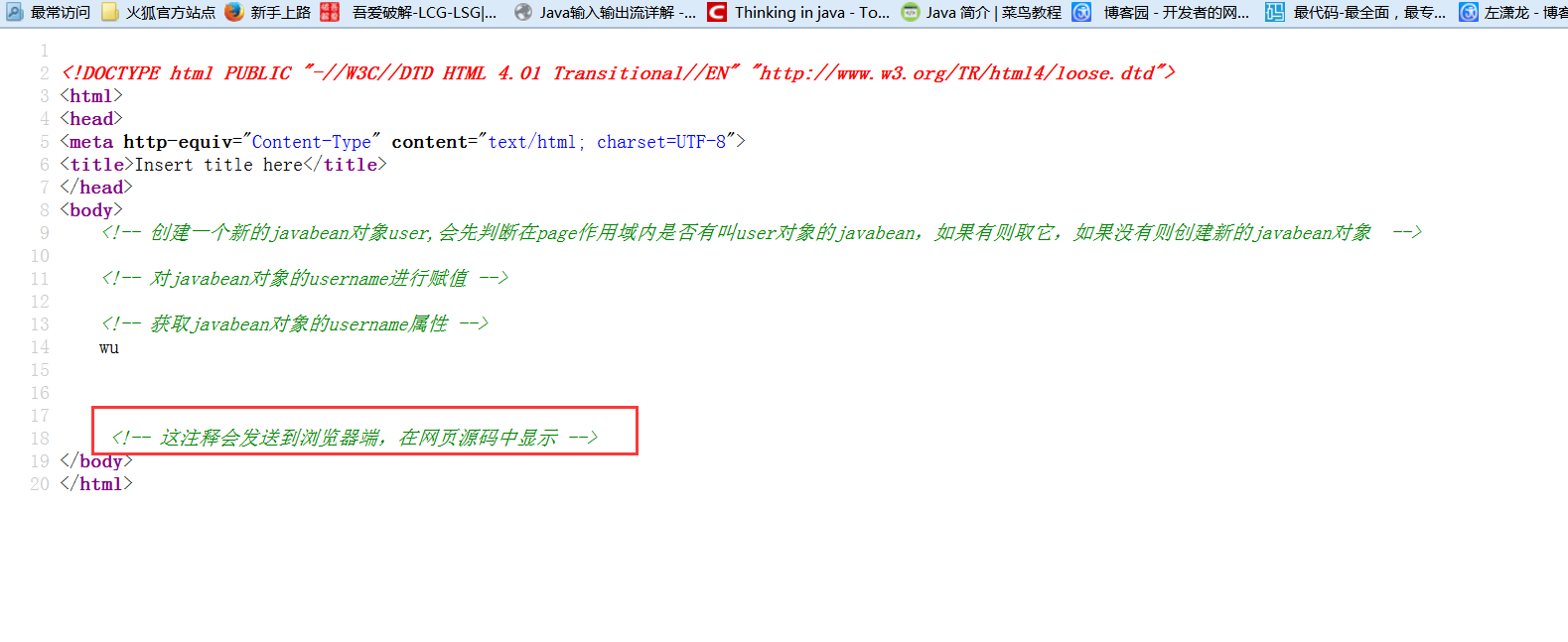
　　　　　　　　　　/\*  \*/　　：Java多行注释

　　　　　　　　　　<!-- --> 　　:这个注释，会发送到浏览器端的源码中显示

　　　　　　　　　　注释分别在servlet中如何显示的？







　　　　　　　　　　JSP注释不会在servlet文件中显示，而java注释则会，但其所有的注释到了浏览器端，都不会出现在源码中，只有<!-- -->这个注释会到浏览器的网页源码中去

　　　　　　4、JSP中申明方法与属性(全局变量)　　使用<%!  方法、属性%>

　　　　　　　　　　这个就不演示了，就是在JSP中编写方法或者属性时，使用<%! %>括起来。

 　　　　　 5、在JSP中使用if语句，或者使用for循环，whilt循环等都可以实现，也就是编写脚本而已。

三、JSP指令

 　　　　指令用来申明JSP页面的一些属性，比如编码方式，文档类型。我们在servlet中也会申明我们使用的编码方式和响应的文档类型的，而JSP就是用指令来申明。上面我们也说到了一条指令，也就是page指令，

　　　　　　　　JSP指令格式：<%@ directive {attribute=value}\* %>

　　　　　　　　　　　解释：

　　　　　　　　　　　　　　directive：指令名称，例如page指令

　　　　　　　　　　　　　　attribute=value：紧跟指令名称后面的就是各种属性，以键值对的形式书写

　　　　　　　　　　　　　　\*：代表后面能跟0个或多个属性。

　　　　　　　　例如： page指令：用来声明JSP页面的属性等。

　　　　　　　　　　　<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>　　page指令，后面跟着三个属性，分别是language、contentType、pageEncoding。这只是其中的几个属性，并没有写全，page指令允许的属性如下表所示

　　　　　　　　属性名称　　　　　　　　取值范围　　　　　　　　　　描述

　　　　　　　　language　　　　　　　　java　　　　　　解释该JSP文件时采用的语言，一般为java语言，默认为java

　　　　　　　　extends　　　　　　　　任何类的全名　　　编译该JSP文件时继承哪个类，JSP为Servlet，因此当指明继承普通类时需要实现Servlet的init、destroy等方法

　　　　　　　　import　　　　　　　　　任何包名、类名　　引入该JSP中用到的类、包等，import是唯一可以声明多次的page指令属性，一个import可以引用uogelei，中间用英文逗号隔开，

　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　如<%@ page import="java.util.List,java.util.ArrayList"%>

　　　　　　　　session　　　　　　　　true、false　　　　该JSP内是否内置Session对象，如果为true，则内置Session对象，可直接使用，否则反之，默认为true

　　　　　　　　autoFlush　　　　　　　true，false　　　　是否运行缓存，如果为true，则使用out.println()等方法输出的字符串并不是立刻到达客户端服务器的，而是暂时存到缓存里，缓存满

　　　　　　　　　　　　　　　　　　　　　　　　　　　　　了或者程序执行完毕或者执行out.flush()操作时才到客户端，默认为true。

　　　　　　　　buffer　　　　　　　　none或者数字KB　　　指定缓存大小，当autoFlush设为true时有效，例如<%@ page buffer=10kb%>

　　　　　　　　isThreadSafe　　　　　　true，false　　　　是否线程安全，如果为true，则运行多个线程同时运行该jsp程序，否则只运行一个线程，其余线程等待，默认为false

　　　　　　　　isErrorPage　　　　　　true，false　　　　　指定该页面是否为错误显示页面，如果为true，则该JSP内置有一个Exception对象exception，可直接使用，否则没有，默认为false

　　　　　　　　errorPage　　　　　某个JSP页面的相对路径　　指明一个错误页面，如果该JSP程序抛出一个未捕捉的异常，则转到errorPage指定的页面，errorPage指定的页面通常

　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　isErrorPage属性为true，且内置的exception对象为未捕捉的异常

　　　　　　　　contentType　　　　　有效的文档类型　　　　客户端浏览器根据该属性判断文档类型，例如 HTML格式为text/html、纯文本格式为text/plain、JPG图像为image/jpeg、GIF图　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　像为image/gif、WORD文档为application/msword，该属性常跟着charset设置编码一起，作用是通知服务器和浏览器都使用同一　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　个码表

　　　　　　　　info　　　　　　　　　　任意字符串　　　　　　指明JSP的信息，该信息可以通过Servlet.getServletInfo()方法获取到

　　　　trimDirective Whitespaces　　　　true、false　　　　　是否去掉指令前后的空白字符，默认为false

　　　　　　　　pageEncoding　　　　UTF-8,ISO-8859-1等　　　　指定一张码表来对该JSP页面进行编码，

　　　　　　include指令

　　　　　　　　　　比较简单，只有一种形式 <%@ include file="relativeURL"%>　　relativeURL：本应用程序内另一个JSP文件或者HTML文件的路径，例如，网址内所有页面均有一个统一风格的导航栏和页脚版权，那么就可以使用该指令将其包含进来，

　　　　　　　　　　特点：include指令会将包含页面的源代码添加到使用include指令的页面中来，然后编译成class文件，而等下会讲到的一个JSP行为，<jsp:include page="relativeURL">作用跟include指令一样，但是不同的是，include行为是运行时单独执行包含页面，然后把执行的结果包含到本页面来，属于先运行后包含。

　　　　　　taglib指令

　　　　　　　　　　JSP支持标签技术，后面会讲到标签的用法，jstl标签库的使用等，

　　　　　　　　　　作用：用来指明JSP页面内使用的JSP标签库，taglib指令有两个属性，uri为类库的地址，prefix为标签的前缀

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

四、JSP行为

 　　　　前面讲了JSP语法，介绍了JSP页面中的内容有哪些，分别有什么作用，就两个东西，模块数据和元素。其中元素有包括脚本，指令，标签，脚本就是JSP中嵌入java代码，指令作用就是申明页面的属性，那标签是干嘛的，标签分为JSP自带内置的标签，和通过taglib指令来使用JSP标签库，或者自定义标签。现在我们先来讲一些JSP内置的标签。

　　　　 JSP内置的标签就被称为JSP行为(JSP Actions)。只要书写很少的标记代码就能使用JSP提供的丰富功能，JSP行为其实是对常用的JSP功能的抽象与封装，可以取代jsp脚本，让JSP中就少一些嵌入java代码的地方。

　　　　　　格式：<jsp:elements ｛attribute="value"｝\* />

　　　　　　　　　　jsp：标签的前缀，说明是jsp内置的标签，

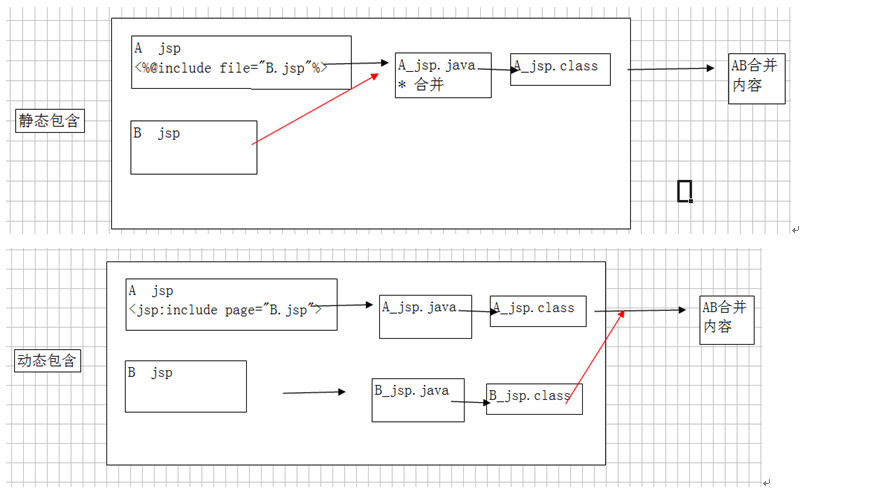
　　　　　　　　　　elements：行为的名称，

　　　　　　　　　　attribute=value：使用键值对来编写属性

　　　　　　　　　　\*：能指定0个或多个属性对

　　　　<jsp:include />行为

　　　　　　　include行为用于运行时包含某个文件，如果被包含的文件为JSP程序，则先会执行JSP程序，然后在把执行的结果包含进来。　作用是跟include指令一样的，唯一的区别就在于，include指令是将被包含的文件的源码加入到了本JSP程序中，然后在进行编译，属于静态包含，而include行为只是将被包含的文件的运行结果包含进自己。属于动态包含



　　　　Java bean行为

　　　　　　　是一组与Java Bean 相关的行为，包括useBean行为、setProperty行为、getProperty行为等

　　　　　　　Java Bean就是普通的Java类，也被称为POJO，只有私有的属性与对应的getter方法和setter方法，注意其中当私有的属性为boolean类型时，习惯上一般把getter方法写成isXxx();而不是getXxx();

　　　　　　　useBean行为

　　　　　　　　　　<jsp:useBean id="beanObject" class="className" scope="Value">　　作用：在jsp中定义一个java bean对象，

　　　　　　　　　　　　　id：指明Java Bean对象的名称，JSP中可以使用该名称引用该Java Bean对象，相当于给new出来的对象取一个变量名，

　　　　　　　　　　　　　class：Java Bean类的全名

　　　　　　　　　　　　　scope：该java bean对象的作用范围，可以写的就四个，也就是JSP的四大作用域，page、request、session、application

　　　　　　　　　　　　　　　　　　page：只能在当前JSP页面使用，如果不在JSP页面，那么就会失效

　　　　　　　　　　　　　　　　　　request：这个前面学过，A页面请求转发到B页面，那么使用的是同一个request，那么A,B页面都算是request的作用域，也就是通过请求转发的页面都是其作　　　　　　　　　　　　　　　　　　　　　　　用域

　　　　　　　　　　　　　　　　　　session：该作用域在一个web项目下任何位置应该读访问的到，只要cookie不关闭，并且cookie设置　　的访问路径为"/"，

　　　　　　　　　　　　　　　　　　application：其实就是Servlet中的servletContext，服务器下的所有项目都能访问到。

　　　　　　　setProperty行为

　　　　　　　　　　<jsp:setProperty name="beanName" property="propertyName" value="">

　　　　　　　　　　　　对Java Bean对象进行属性的设置

　　　　　　　　　　　　name：java bean对象的名称，也就是在useBean行为中的id

　　　　　　　　　　　　property：对象中的属性名，

　　　　　　　　　　　　value：要对其属性进行赋值的值

　　　　　　　getProperty行为

　　　　　　　　　　<jsp:getProperty name="beanName" property="propertyName" />

　　　　　　　　　　　　获取JavaBean对象的某个属性值

　　　　　　　　　　　　name:java bean 对象的名称，也就是在useBean行为中的id

　　　　　　　　　　　　property：对象的属性名

　　　　　　例子：　　javabean：User.java　　NewFile.jsp

　　　　　　User.java

IMG_264 View Code

　　　　　　NewFile.jsp

IMG_265 View Code

　　　　　　可以查看NewFile.jsp变为servlet后的源代码，看看我们写的javabean行为会被转换为何种语句



　　　　　　这里出现了一个JSP九大内置对象中的一个，pageContext。现在简单提一句，pageContext就是JSP页面的管理者(上下文)，其中的getAttribute(name，scope)方法是获取指定作用域中的数据的，如果getAttribute(name)方法的话，默认是对page作用域进行操作，findAttribute(name)依次从page、request、session、application获得内容。

　　　　　　在第一个红框中，就代表中我们的useBean行为，其中进行了一次判断，就是如果在page作用域中找不到user这个对象，那么就创建一个新的，否则就使用找到的这个user对象，

　　　　　　第二个红框中，代表着我们的setProperty行为，先找到user对象，然后在对其属性进行赋值

　　　　　　第三个红框中，代表着我们的getProperty行为，也是先找到user对象，然后在获取其属性的值。

　　　　　　注意：对于javabean行为来说，有一个特点的地方，就是当请求过来的参数对应javabean的属性时，可以为其一次性设置所有的值

　　　　　　　　　　<jsp:setProperty name="user" property="\*" /> //设置user的所有属性，属性值从request中自动取得，\*代表所有属性。

　　　　<jsp:forward />行为

　　　　　　实现请求转发功能，Servlet中通过request.getRequestDispatcher("someServlet").forward(request,response);而在JSP中也能够实现相同的功能，只不过用的是<jsp:forward />行为，实际上forward行为就是对其进行了封装。

　　　　　　格式：

　　　　　　　　<jsp:forward page="someServlet">

　　　　　　　　　　　<jsp:param name="param1" value="value1"/>

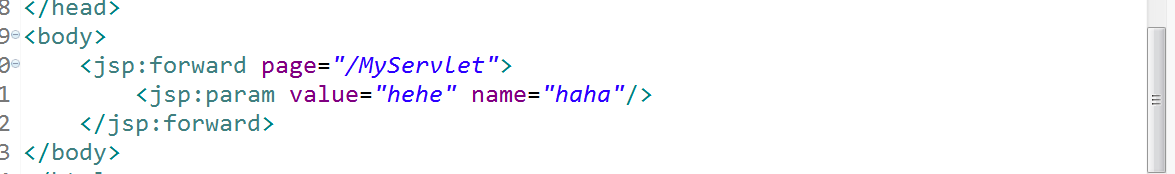
　　　　　　　　　　　<jsp:param name="param2" value="value2"/>

　　　　　　　　</jsp:forward>

　　　　　　　　page:需要跳转到的页面或者servlet、　<jsp:param/>参数行为，带一些参数过去，name、value是以键值对的形式带过去的

　　　　　　例如：

　　　　　　　　　NewFile.jsp



　　　　　　　　　MyServlet.java

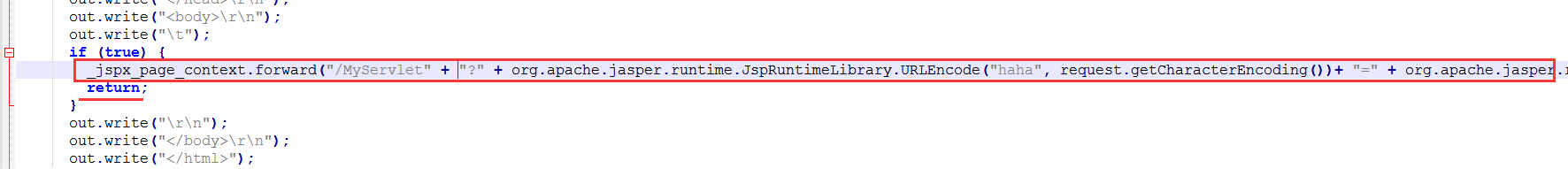
IMG_268

　　　　　　　　　访问：http://localhost:8080/test01/NewFile.jsp



　　　　　　　　　　浏览器地址栏没有改变，说明是请求转发

　　　　　　　　　NewFile\_jsp.java



　　　　　　　　　　使用return的好处是执行完上面的转发，就直接return，没有必要在执行下面的代码了，对参数使用了URLEncode进行编码，说明该<jsp:param/>可以直接传递中文，但是前提是要设置request.setCharacterEncoding("UTF-8");为什么这样做，看上面框起来中的代码。

　　　　<jsp:directive/>行为

　　　　　　　directive行为，就相当于JSP指令，比如<jsp:directive.page/>相当于<%@ page %>指令，等等其它指令是一样的书写格式。

五、JSP隐藏的九大内置对象和四大作用域

 　　　　之前那么一大段内容，就是为了讲解什么是JSP，JSP中的内容是怎么样的，到这里应该知道的差不多了， 但我们还要深入了解一些东西，我们知道JSP中的内容就只有两种，模版数据，和元素，元素就包括了指令呀，脚本呀，标签(行为)呀，脚本会慢慢被标签全部代替，也就是说JSP中基本上不会嵌入Java代码，但是我们也知道JSP会转换为servlet，在Servlet中，输出数据时，都需要通过response.getWrite()；但是在JSP中，直接使用out对象进行输出，为什么呢？这就是因为out为JSP的一个隐藏对象，JSP中内置了9个隐藏对象，使得JSP比Servlet使用起来更简单，更方便，

　　　　page、config、application、request、response、session、out、exception、pageContext

　　　　　　page：page对象代表当前JSP页面，是当前JSP编译后的Servlet类的对象。相当于this。

　　　　　　config：标识Servlet配置，类型：ServletConfig，api跟Servlet中的ServletConfig对象是一样的，能获取该servlet的一些配置信息，能够获取ServletContext

　　　　　　application：标识web应用上下文，类型：ServletContext，详情就看Servlet中的ServletContext的使用

　　　　　　request:请求对象，　　类型：httpServletRequest

　　　　　　response:响应对象　　类型：httpServletResponse

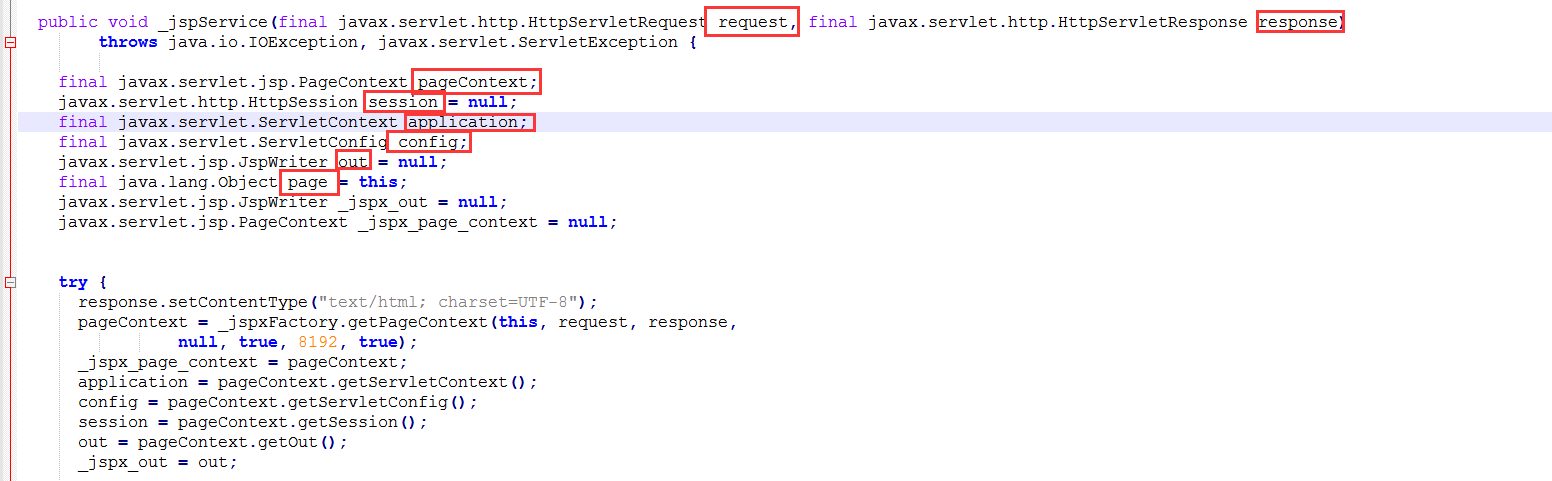
　　　　　　session：表示一次会话，在服务器端记录用户状信息的技术

　　　　　　out：输出响应体 类型：JspWriter

　　　　　　exception 表示发生异常对象，类型 Throwable，在上面我们介绍page指令中的一个errorPage属性时就有说到他

　　　　　　pageContext：表示 jsp页面上下文（jsp管理者） 类型：PageContext

　　　　　标记了红色的对象就是JSP独有的，其他的都是Servlet中的老东西。



　　　　　　在这个由jsp转换为servlet的文件中，只能看到8个内置对象，少了exception对象，因为我们在将page指令时，说过一个isErrorPage属性，默认是false，被关闭了，所以其中并没有exception对象。

　　　　　JSP的四大作用域：page、request、session、application

　　　　　　 　这四大作用域，其实就是其九大内置对象中的四个，为什么说他们也是JSP的四大作用域呢？因为这四个对象都能存储数据，比如request.setAttribute()注意和request.setParameter()区分开来，一个是存储在域中的、一个是请求参数，session.setAttribute()、application其实就是SerlvetContext，自然也有setAttribute()方法。而page作用域的操作就需要依靠pageContext对象来进行了。在上面我们也有提到JSP的四大作用域，

　　　　　　　　page作用域：代表变量只能在当前页面上生效

　　　　　　　　request：代表变量能在一次请求中生效，一次请求可能包含一个页面，也可能包含多个页面，比如页面A请求转发到页面B

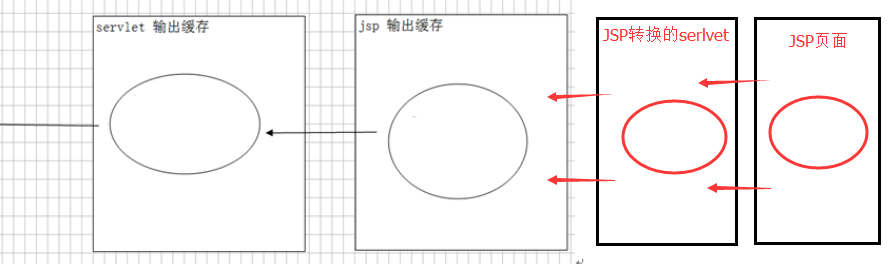
　　　　　　　　session：代表变量能在一次会话中生效，基本上就是能在web项目下都有效，session的使用也跟cookie有很大的关系。一般来说，只要浏览器不关闭，cookie就会一直生效，cookie生效，session的使用就不会受到影响。

　　　　　　　　application：代表变量能一个应用下(多个会话)，在服务器下的多个项目之间都能够使用。比如baidu、wenku等共享帐号。

　　　　　out对象：

　　　　　　　　类型：JspWriter

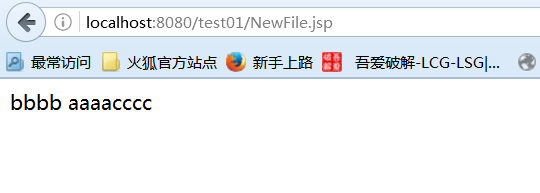
　　　　　　　　jsp 输出底层使用 response.getWriter()；什么意思呢？这里就要讲解一下JSP缓存和Servlet缓存了，输出的过程是这样的



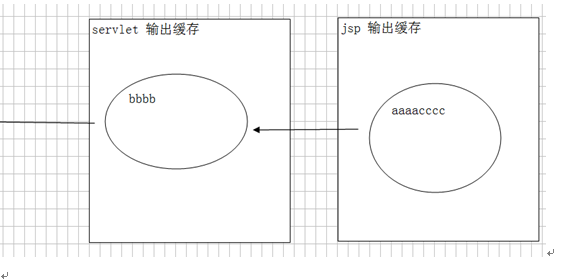
　　　　　　　　JSP页面转换为Servlet后，使用的out对象是JspWriter类型的，所以是会先将要发送的数据存入JSP输出缓存中，然后，等JSP输出缓存满了在自动刷新到servlet输出缓存，等serlvet输出缓存满了，或者程序结束了，就会将其输出到浏览器上。除非手动out.flush()。

　　　　　　　　验证servlet输出缓存和JSP输出缓存和我们上面所说的是正确的。





　　　　　　　　　　分析：如果按没有jsp缓存和servlet缓存的话，输出的结果应该是aaaabbbbcccc，但是输出的却是bbbbaaaacccc，为什么呢？按照我们上面所说的原理进行分析，out对象是先将其输出到JSP缓存中，所以aaaa加入了jsp缓存，而response.getWriter().print("bbbb")是直接将bbbb输出到servlet缓存中，然后又使用out对象将cccc输出到jsp缓存，到程序结束，servlet缓存中有bbbb，然后jsp会将缓存中的内容就刷新到servlet缓存中，serlvet就是bbbbaaaacccc了，然后到浏览器也就得到我们的输出结果了。如果在12行将注释去掉，那么输出的结果又会是什么呢？答案就是aaaabbbbcccc，过程自行分析。



　　　　　pageContext对象：重点

　　　　　　　　这个功能就比较强大了，比较牛逼，基本上什么他都有，因为是它是JSP页面的管理者(上下文)，所以JSP中的内置对象呀，它统统能够获得，下面介绍它的api。

　　　　　　　　1、获得其它八大内置对象 getXxx()

　　　　　　　　　　pageContext.getOut();　　//获得out对象

　　　　　　　　　　pageContext.getApplication();　　//获得application对象

　　　　　　　　　　等等....

　　　　　　　　2、对作用域的属性进行操作(四大作用域)

　　　　　　　　　　对默认作用域的属性进行操作。page

　　　　　　　　　　pageContext.getAttribute(name);　　//获得page作用域数据

　　　　　　　　　　pageContext.setAttribute(name,value);　　//给page作用域设置内容

　　　　　　　　　　pageContext.removeAttribute(name);　　//给page作用域移除内容

　　　　　　　　3、对指定作用域的属性进行操作

　　　　　　　　　　getAttribute(name,scope);　　//获得 指定作用域中的数据

　　　　　　　　　　setAttribute(name,value);　　//给指定作用域设置内容

　　　　　　　　　　removeAttribute(name ,scope)  移除指定作用域的内容（page/request/session/application）

　　　　　　　　4、提供作用域常量

　　　　　　　　　　PageContext.PAGE\_SCOPE　　page

　　　　　　　　　　PageContext.REQUEST\_SCOPE　　request

　　　　　　　　　　PageContext.SESSION\_SCOPE　　response

　　　　　　　　　　PageContext.APPLICATION\_SCOPE　　application

　　　　　　　　5、一次获得指定名称内容

　　　　　　　　　　findAttribute(name);　　//依次从page、request、session、application 获得内容

　　　　　response对象：

　　　　　　　　就是响应对象，、如果不了解就看看讲解request和response的这一章节的内容

　　　　　config对象：

　　　　　　　　类型：ServletConfig

　　　　　　　　能够获取servlet的初始化参数，获取servletContext对象，获取servletName

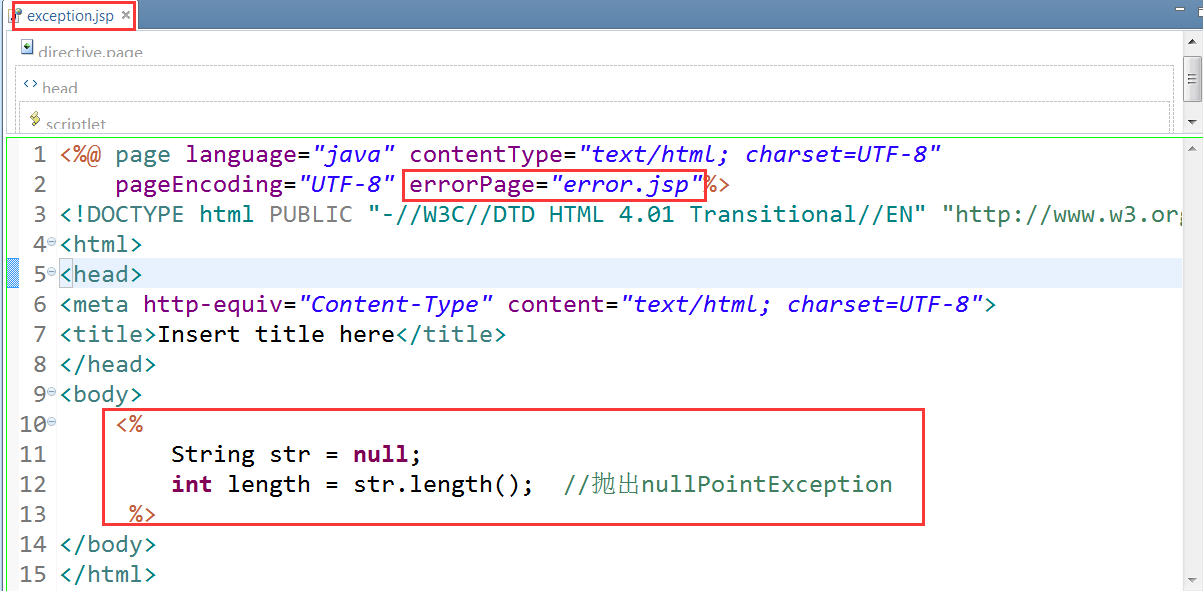
　　　　　　　　api详情请看讲解servlet这一章节

　　　　　exception异常对象：

　　　　　　　　包含了异常的信息

　　　　　　　　使用它，必须结合page指令中的isErrorPage属性和errorPage属性。

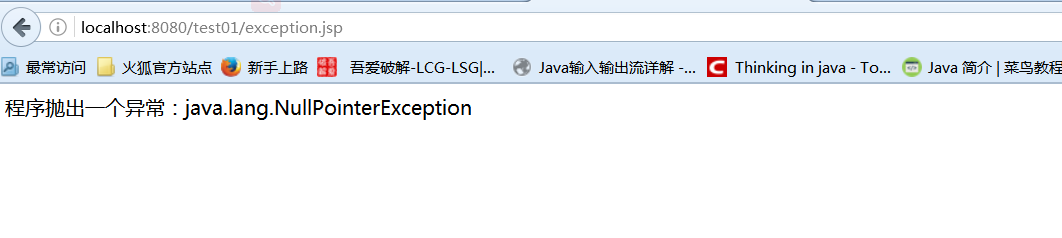
　　　　　　　　exception.jsp　　抛异常的一个NullPointException，并且跳转到error.jsp错误显示页面　　errorPage属性的意思是如果发生未捕捉到的异常，将会跳转到error.jsp页面



　　　　　　　　error.jsp　　isErrorPage属性说明该页面是一个错误显示页面，则可以使用exception对象



　　　　　　　　访问http://localhost:8080/test01/exception.jsp



六、总结

　　　　通过上面一大堆一大堆的学习，我们应该知道这些东西

　　　　1、什么是JSP？

　　　　　　JSP本质上就是一个servlet，因为servlet输出html太麻烦了，所以就有了JSP，JSP就是专门用来书写html的，当然其中也能写java代码。

　　　　2、JSP的内容包括什么？

　　　　　　模版数据和元素。其中元素有包括脚本(java代码)、指令(页面属性)、和行为(标签，为了JSP中不嵌入那么多java代码衍生的)

　　　　3、JSP中九大内置对象是哪九个？

　　　　　　九大内置对象，page、config、appliction、request、response、session、out、exception、pageContext

　　　　4、九大内置对象和servlet中对象的关系

　　　　　　page就是jsp转换为servletservlet对象本身，也就是this

　　　　　　config -- Servlet中的servletConfig

　　　　　　application -- Servlet中的ServletContext

　　　　　　request　　-- Servlet中的request

　　　　　　response　　-- Servlet中的response

　　　　　　session　　-- Servlet中的session

　　　　　　out　　--　JspWriter

　　　　　　exception　　-- 异常对象

　　　　　　pageContext　　-- 表示 jsp页面上下文（jsp管理者） 类型：PageContext，

　　　　　　　　其中pageContext对象最牛逼，有了他就拥有了天下，哈哈~

　　　5、JSP中的四大作用域。

　　　　　　page、request、session、application

　　　　　　其中操作page域中属性需要借助pageContext对象。

## **一、EL表达式简介**

　　EL 全名为Expression Language。EL主要作用：  
****1、获取数据****  
　　　　EL表达式主要用于替换JSP页面中的脚本表达式，以从各种类型的web域 中检索java对象、获取数据。(某个web域 中的对象，访问javabean的属性、访问list集合、访问map集合、访问数组)  
****2、执行运算****  
　　　　利用EL表达式可以在JSP页面中执行一些基本的关系运算、逻辑运算和算术运算，以在JSP页面中完成一些简单的逻辑运算。${user==null}  
****3、获取web开发常用对象****  
　　　　EL 表达式定义了一些隐式对象，利用这些隐式对象，web开发人员可以很轻松获得对web常用对象的引用，从而获得这些对象中的数据。  
****4、调用Java方法****  
　　　　EL表达式允许用户开发自定义EL函数，以在JSP页面中通过EL表达式调用Java类的方法。

### **1.1、获取数据**

　　使用EL表达式获取数据语法："****${标识符}****"  
　　EL表达式语句在执行时，会调用pageContext.findAttribute方法，用标识符为关键字，分别从page、request、session、application四个域中查找相应的对象，找到则返回相应对象，找不到则返回”” （注意，不是null，而是空字符串）。

　　EL表达式可以很轻松获取JavaBean的属性，或获取数组、Collection、Map类型集合的数据

****el表达式获取数据范例：****

IMG_279

1 <%@ page language="java" import="java.util.\*" pageEncoding="UTF-8"%>

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

3 <%@page import="me.gacl.domain.Person"%>

4 <%@page import="me.gacl.domain.Address"%>

5 <!DOCTYPE HTML>

6 <html>

7 <head>

8 <title>el表达式获取数据</title>

9 </head>

10

11 <body>

12 <%

13 request.setAttribute("name","孤傲苍狼");

14 %>

15 <%--${name}等同于pageContext.findAttribute("name") --%>

16 使用EL表达式获取数据：${name}

17 <hr>

18 <!-- 在jsp页面中，使用el表达式可以获取bean的属性 -->

19 <%

20 Person p = new Person();

21 p.setAge(12);

22 request.setAttribute("person",p);

23 %>

24 使用el表达式可以获取bean的属性：${person.age}

25 <hr>

26 <!-- 在jsp页面中，使用el表达式可以获取bean中的。。。。。。。。。的属性 -->

27 <%

28 Person person = new Person();

29 Address address = new Address();

30 person.setAddress(address);

31

32 request.setAttribute("person",person);

33 %>

34 ${person.address.name}

35 <hr>

36 <!-- 在jsp页面中，使用el表达式获取list集合中指定位置的数据 -->

37 <%

38 Person p1 = new Person();

39 p1.setName("孤傲苍狼");

40

41 Person p2 = new Person();

42 p2.setName("白虎神皇");

43

44 List<Person> list = new ArrayList<Person>();

45 list.add(p1);

46 list.add(p2);

47

48 request.setAttribute("list",list);

49 %>

50

51 <!-- 取list指定位置的数据 -->

52 ${list[1].name}

53

54 <!-- 迭代List集合 -->

55 <c:forEach var="person" items="${list}">

56 ${person.name}

57 </c:forEach>

58 <hr>

59 <!-- 在jsp页面中，使用el表达式获取map集合的数据 -->

60 <%

61 Map<String,String> map = new LinkedHashMap<String,String>();

62 map.put("a","aaaaxxx");

63 map.put("b","bbbb");

64 map.put("c","cccc");

65 map.put("1","aaaa1111");

66 request.setAttribute("map",map);

67 %>

68

69 <!-- 根据关键字取map集合的数据 -->

70 ${map.c}

71 ${map["1"]}

72 <hr>

73 <!-- 迭代Map集合 -->

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

75 ${me.key}=${me.value}<br/>

76 </c:forEach>

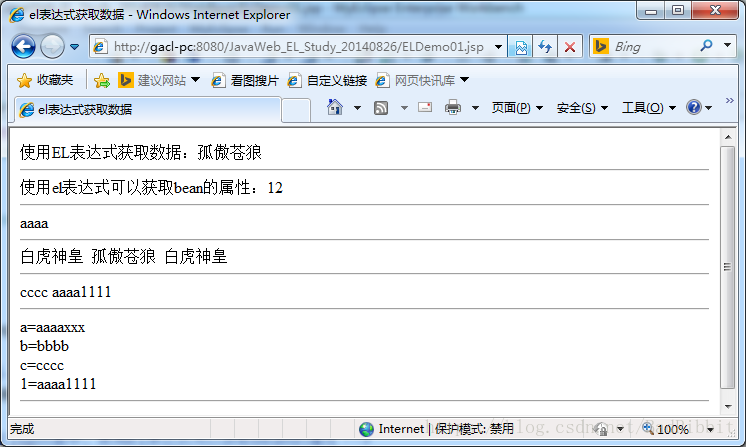
77 <hr>

78 </body>

79 </html>

IMG_280

　　运行效果如下：



### **1.2、执行运算**

　　语法：${运算表达式}，EL表达式支持如下运算符：

　　1、关系运算符



　　2、逻辑运算符：



****3、empty运算符****：检查对象是否为null(空)

****4、二元表达式****：${user!=null?user.name :""}  
　　****5、[ ] 和 . 号运算符****  
  
****使用EL表达式执行运算范例：****

IMG_284

1 <%@ page language="java" import="java.util.\*" pageEncoding="UTF-8"%>

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

3 <%@page import="me.gacl.domain.User"%>

4 <!DOCTYPE HTML>

5 <html>

6 <head>

7 <title>el表达式运算符</title>

8 </head>

9

10 <body>

11 <h3>el表达式进行四则运算：</h3>

12 加法运算：${365+24}<br/>

13 减法运算：${365-24}<br/>

14 乘法运算：${365\*24}<br/>

15 除法运算：${365/24}<br/>

16

17 <h3>el表达式进行关系运算：</h3>

18 <%--${user == null}和 ${user eq null}两种写法等价--%>

19 ${user == null}<br/>

20 ${user eq null}<br/>

21

22 <h3>el表达式使用empty运算符检查对象是否为null(空)</h3>

23 <%

24

25 List<String> list = new ArrayList<String>();

26 list.add("gacl");

27 list.add("xdp");

28 request.setAttribute("list",list);

29 %>

30 <%--使用empty运算符检查对象是否为null(空) --%>

31 <c:if test="${!empty(list)}">

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

33 ${str}<br/>

34 </c:forEach>

35 </c:if>

36 <br/>

37 <%

38 List<String> emptyList = null;

39 %>

40 <%--使用empty运算符检查对象是否为null(空) --%>

41 <c:if test="${empty(emptyList)}">

42 对不起，没有您想看的数据

43 </c:if>

44

45 <br/>

46

47 <h3>EL表达式中使用二元表达式</h3>

48 <%

49 session.setAttribute("user",new User("孤傲苍狼"));

50 %>

51 ${user==null? "对不起，您没有登陆 " : user.username}

52

53 <br/>

54

55 <h3>EL表达式数据回显</h3>

56 <%

57 User user = new User();

58 user.setGender("male");

59 //数据回显

60 request.setAttribute("user",user);

61 %>

62 <input type="radio" name="gender" value="male" ${user.gender=='male'?'checked':''}>男

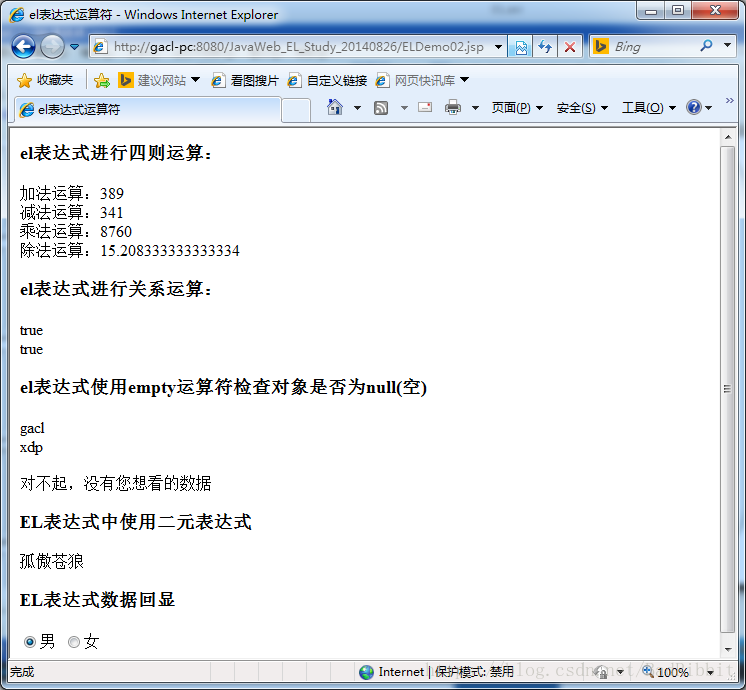
63 <input type="radio" name="gender" value="female" ${user.gender=='female'?'checked':''}>女

64 <br/>65 </body>

66 </html>

IMG_285

　　运行结果如下：



### **1.3、获得web开发常用对象**

　　EL表达式语言中定义了11个隐含对象，使用这些隐含对象可以很方便地获取web开发中的一些常见对象，并读取这些对象的数据。  
　　语法：****${隐式对象名称}****：获得对象的引用

|  |  |  |
| --- | --- | --- |
| 序号 | 隐含对象名称 | 描       述 |
| 1 | pageContext | 对应于JSP页面中的pageContext对象（注意：取的是pageContext对象。） |
| 2 | pageScope | 代表page域中用于保存属性的Map对象 |
| 3 | requestScope | 代表request域中用于保存属性的Map对象 |
| 4 | sessionScope | 代表session域中用于保存属性的Map对象 |
| 5 | applicationScope | 代表application域中用于保存属性的Map对象 |
| 6 | param | 表示一个保存了所有请求参数的Map对象 |
| 7 | paramValues | 表示一个保存了所有请求参数的Map对象，它对于某个请求参数，返回的是一个string[] |
| 8 | header | 表示一个保存了所有http请求头字段的Map对象，注意：如果头里面有“-” ，例Accept-Encoding，则要header[“Accept-Encoding”] |
| 9 | headerValues | 表示一个保存了所有http请求头字段的Map对象，它对于某个请求参数，返回的是一个string[]数组。注意：如果头里面有“-” ，例Accept-Encoding，则要headerValues[“Accept-Encoding”] |
| 10 | cookie | 表示一个保存了所有cookie的Map对象 |
| 11 | initParam | 表示一个保存了所有web应用初始化参数的map对象 |

 测试EL表达式中的11个隐式对象：

IMG_287

1 <%@ page language="java" import="java.util.\*" pageEncoding="UTF-8"%>

2 <!DOCTYPE HTML>

3 <html>

4 <head>

5 <title>el隐式对象</title>

6 </head>

7

8 <body>

9 <br/>---------------1、pageContext对象：获取JSP页面中的pageContext对象------------------------<br/>

10 ${pageContext}

11 <br/>---------------2、pageScope对象：从page域(pageScope)中查找数据------------------------<br/>

12 <%

13 pageContext.setAttribute("name","孤傲苍狼"); //map

14 %>

15 ${pageScope.name}

16 <br/>---------------3、requestScope对象：从request域(requestScope)中获取数据------------------------<br/>

17 <%

18 request.setAttribute("name","白虎神皇"); //map

19 %>

20 ${requestScope.name}

21 <br/>---------------4、sessionScope对象：从session域(sessionScope)中获取数据------------------------<br/>

22 <%

23 session.setAttribute("user","xdp"); //map

24 %>

25 ${sessionScope.user}

26 <br/>---------------5、applicationScope对象：从application域(applicationScope)中获取数据------------------------<br/>

27 <%

28 application.setAttribute("user","gacl"); //map

29 %>

30 ${applicationScope.user}

31 <br/>--------------6、param对象：获得用于保存请求参数map，并从map中获取数据------------------------<br/>

32 <!-- http://localhost:8080/JavaWeb\_EL\_Study\_20140826/ELDemo03.jsp?name=aaa -->

33 ${param.name}

34 <!-- 此表达式会经常用在数据回显上 -->

35 <form action="${pageContext.request.contextPath}/servlet/RegisterServlet" method="post">

36 <input type="text" name="username" value="${param.username}">

37 <input type="submit" value="注册">

38 </form>

39 <br/>--------------7、paramValues对象：paramValues获得请求参数 //map{"",String[]}------------------------<br/>

40 <!-- http://localhost:8080/JavaWeb\_EL\_Study\_20140826/ELDemo03.jsp?like=aaa&like=bbb -->

41 ${paramValues.like[0]}

42 ${paramValues.like[1]}

43 <br/>--------------8、header对象：header获得请求头------------------------<br/>

44 ${header.Accept}<br/>

45 <%--${header.Accept-Encoding} 这样写会报错

46 测试headerValues时，如果头里面有“-” ，例Accept-Encoding，则要headerValues[“Accept-Encoding”]

47 --%>

48 ${header["Accept-Encoding"]}

49 <br/>--------------9、headerValues对象：headerValues获得请求头的值------------------------<br/>

50 <%--headerValues表示一个保存了所有http请求头字段的Map对象，它对于某个请求参数，返回的是一个string[]数组

51 例如：headerValues.Accept返回的是一个string[]数组 ，headerValues.Accept[0]取出数组中的第一个值

52 --%>

53 ${headerValues.Accept[0]}<br/>

54 <%--${headerValues.Accept-Encoding} 这样写会报错

55 测试headerValues时，如果头里面有“-” ，例Accept-Encoding，则要headerValues[“Accept-Encoding”]

56 headerValues["Accept-Encoding"]返回的是一个string[]数组，headerValues["Accept-Encoding"][0]取出数组中的第一个值

57 --%>

58 ${headerValues["Accept-Encoding"][0]}

59 <br/>--------------10、cookie对象：cookie对象获取客户机提交的cookie------------------------<br/>

60 <!-- 从cookie隐式对象中根据名称获取到的是cookie对象,要想获取值，还需要.value -->

61 ${cookie.JSESSIONID.value} //保存所有cookie的map

62 <br/>--------------11、initParam对象：initParam对象获取在web.xml文件中配置的初始化参数------------------------<br/>

63 <%--

64 <!-- web.xml文件中配置初始化参数 -->

65 <context-param>

66 <param-name>xxx</param-name>

67 <param-value>yyyy</param-value>

68 </context-param>

69 <context-param>

70 <param-name>root</param-name>

71 <param-value>/JavaWeb\_EL\_Study\_20140826</param-value>

72 </context-param>

73 --%>

74 <%--获取servletContext中用于保存初始化参数的map --%>

75 ${initParam.xxx}<br/>

76 ${initParam.root}

77 </body>

78 </html>

IMG_288

RegisterServlet的代码如下：

IMG_289

1 package me.gacl.web.controller;

2

3 import java.io.IOException;

4 import javax.servlet.ServletException;

5 import javax.servlet.http.HttpServlet;

6 import javax.servlet.http.HttpServletRequest;

7 import javax.servlet.http.HttpServletResponse;

8

9 public class RegisterServlet extends HttpServlet {

10 /\*

11 \* 处理用户注册的方法

12 \*/

13 public void doGet(HttpServletRequest request, HttpServletResponse response)

14 throws ServletException, IOException {

15 //1、接收参数

16 String userName = request.getParameter("username");

17 /\*\*

18 \* 2、直接跳转回/ELDemo03.jsp页面，没有使用request.setAttribute("userName", userName)将userName存储到request对象中

19 \* 但是在ELDemo03.jsp页面中可以使用${param.username}获取到request对象中的username参数的值

20 \*/

21 request.getRequestDispatcher("/ELDemo03.jsp").forward(request, response);

22 }

23

24 public void doPost(HttpServletRequest request, HttpServletResponse response)

25 throws ServletException, IOException {

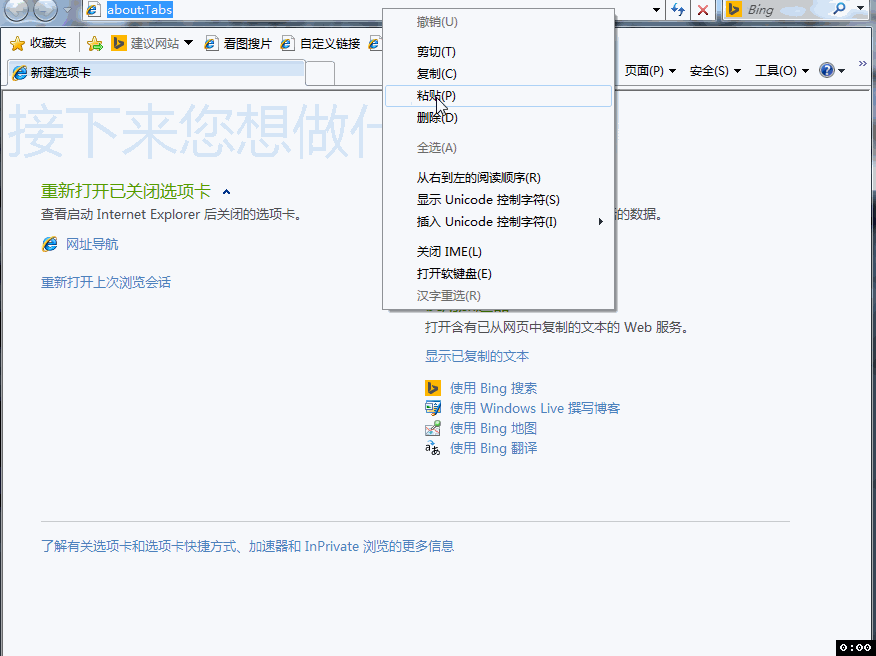
26 doGet(request, response);

27 }

28 }

IMG_290

测试结果如下：



注意:  
　　****测试header和headerValues时，如果头里面有“-” ，例Accept-Encoding，则要header["Accept-Encoding"]、headerValues["Accept-Encoding"]****  
****测试cookie时，例${cookie.key}取的是cookie对象，如访问cookie的名称和值，须${cookie.key.name}或${cookie.key.value}****

### **1.4、使用EL调用Java方法**

　　EL表达式语法允许开发人员开发自定义函数，以调用Java类的方法。语法：${prefix：method(params)}  
　　在EL表达式中调用的只能是Java类的静态方法，这个Java类的静态方法需要在TLD文件中描述，才可以被EL表达式调用。  
　　EL自定义函数用于扩展EL表达式的功能，可以让EL表达式完成普通Java程序代码所能完成的功能。

### **1.5、EL Function开发步骤**

　　一般来说， EL自定义函数开发与应用包括以下三个步骤：  
　　1、编写一个Java类的静态方法  
　　2、编写标签库描述符（tld）文件，在tld文件中描述自定义函数。  
　　3、在JSP页面中导入和使用自定义函数

示例：开发对html标签进行转义的el function

1、编写html转义处理工具类，工具类中添加对html标签进行转义的静态处理方法，如下：

IMG_292

1 package me.gacl.util;

2

3 /\*\*

4 \* @ClassName: HtmlFilter

5 \* @Description: html转义处理工具类

6 \* @author: 孤傲苍狼

7 \* @date: 2014-8-27 上午12:09:15

8 \*

9 \*/

10 public class HtmlFilter {

11

12 /\*\*

13 \* @Method: filter

14 \* @Description: 静态方法，html标签转义处理

15 \* @Anthor:孤傲苍狼

16 \*

17 \* @param message 要转义的内容

18 \* @return 转义后的内容

19 \*/

20 public static String filter(String message) {

21

22 if (message == null)

23 return (null);

24

25 char content[] = new char[message.length()];

26 message.getChars(0, message.length(), content, 0);

27 StringBuffer result = new StringBuffer(content.length + 50);

28 for (int i = 0; i < content.length; i++) {

29 switch (content[i]) {

30 case '<':

31 result.append("&lt;");

32 break;

33 case '>':

34 result.append("&gt;");

35 break;

36 case '&':

37 result.append("&amp;");

38 break;

39 case '"':

40 result.append("&quot;");

41 break;

42 default:

43 result.append(content[i]);

44 }

45 }

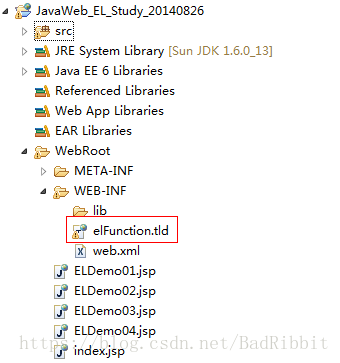
46 return (result.toString());

47 }

48 }

IMG_293

2、在WEB-INF目录下编写标签库描述符（tld）文件，在tld文件中描述自定义函数



　　elFunction.tld的代码如下：

IMG_295

1 <?xml version="1.0" encoding="UTF-8"?>

2 <taglib version="2.0" xmlns="http://java.sun.com/xml/ns/j2ee"

3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee web-jsptaglibrary\_2\_0.xsd">

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

5 <short-name>EL Function</short-name>

6 <!--

7 自定义EL函数库的引用URI，

8 在JSP页面中可以这样引用：<%@taglib uri="/ELFunction" prefix="fn" %>

9 -->

10 <uri>/ELFunction</uri>

11

12 <!--<function>元素用于描述一个EL自定义函数 -->

13 <function>

14 <description>html标签转义处理方法</description>

15 <!--<name>子元素用于指定EL自定义函数的名称-->

16 <name>filter</name>

17 <!--<function-class>子元素用于指定完整的Java类名-->

18 <function-class>me.gacl.util.HtmlFilter</function-class>

19 <!--<function-signature>子元素用于指定Java类中的静态方法的签名，

20 方法签名必须指明方法的返回值类型及各个参数的类型，各个参数之间用逗号分隔。-->

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

22 </function>

23

24 </taglib>

IMG_296

3、在JSP页面中导入和使用自定义函数

IMG_297

1 <%@ page language="java" import="java.util.\*" pageEncoding="UTF-8"%>

2 <%--引入EL自定义函数库 --%>

3 <%@taglib uri="/ELFunction" prefix="fn" %>

4 <!DOCTYPE HTML>

5 <html>

6 <head>

7 <title>使用EL调用Java方法</title>

8 </head>

9

10 <body>

11 <%--使用EL调用filter方法--%>

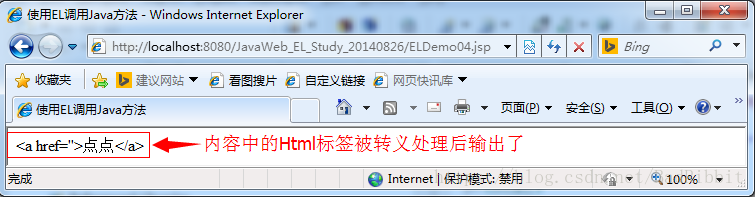
12 ${fn:filter("<a href=''>点点</a>")}

13 </body>

14 </html>

IMG_298

运行结果如下：



### **1.6、开发EL Function注意事项**

　　编写完标签库描述文件后，需要将它放置到<web应用>\WEB-INF目录中或WEB-INF目录下的除了classes和lib目录之外的任意子目录中。   
　　TLD文件中的<uri> 元素用指定该TLD文件的URI，在JSP文件中需要通过这个URI来引入该标签库描述文件。  
　　<function>元素用于描述一个EL自定义函数，其中：  
　　<name>子元素用于指定EL自定义函数的名称。  
　　<function-class>子元素用于指定完整的Java类名，  
　　<function-signature>子元素用于指定Java类中的静态方法的签名，方法签名必须指明方法的返回值类型及各个参数的类型，各个参数之间用逗号分隔。

### **1.7、EL注意事项**

　　EL表达式是JSP 2.0规范中的一门技术 。因此，若想正确解析EL表达式，需使用支持Servlet2.4/JSP2.0技术的WEB服务器。  
注意：有些Tomcat服务器如不能使用EL表达式  
    （1）升级成tomcat6  
    （2）在JSP中加入<%@ page isELIgnored="false" %>

### **1.8、EL表达式保留关键字**



　　所谓保留字的意思是指变量在命名时，应该避开上述的名字，以免程序编译时发生错误，关于EL表达式的内容的总结就这么多。

二、JSTL标签库

　　　　JSTL标签库中包括很多种类的标签core、fmt、fn方法库、sql标签库，使用JSTL标签库，需要用taglib指令导入JSTL标签库

　　　　core标签库

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

　　　　　　讲解<c:out/>、<c:set/>、<c:remove/>、<c:if  test=""/>、<c:choose  />、<c:forTokens />、<c:catch /> 、<c:url  />、 <c:redirect />、<c:forEach/>、

　　　　<c:out/>标签

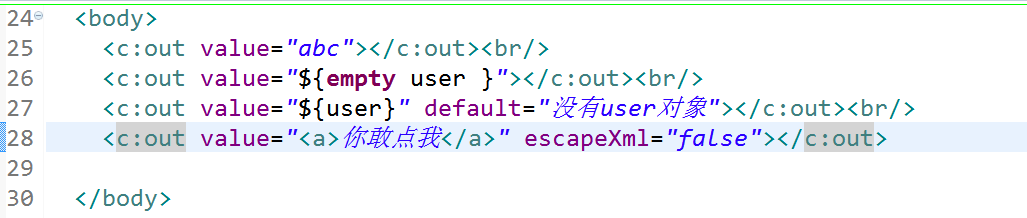
　　　　　　out ：将指定内容输出到浏览器，默认支持el表达式,

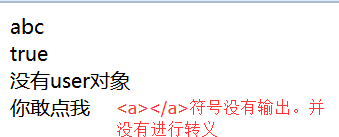
　　　　　　<c:out value="" default="" escapeXml />

　　　　　　　　　　value:输出的值，可以为EL表达式

　　　　　　　　　　default：如果value属性的值不存在，会输出default属性值

　　　　　　　　　　excapeXml：如果为true，则会为value中的特殊字符进行转义，默认为true





　　　　　　<c:set/>和<c:remove/>标签



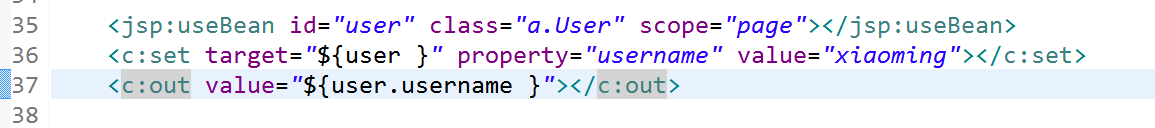
　　　　　　　　　　图中只介绍set标签的一种用法，其中var不能接受EL表达式，只能是字符，

　　　　　　　　　　<c:set target="" property="" value="" />

　　　　　　　　　　　　target:跟var类似，但是它能够接受EL表达式，可以是一个对象，一个map等，和var进行互补

　　　　　　　　　　　　property:对象中的属性名

　　　　　　　　　　　　value:属性赋值



　　　　　　　　　　使用哪种set标签，就看自己的需求了。

　　　　　　<c:if  test=""/>标签

　　　　　　　　　　　　if条件语句



　　　　　　<c:choose  />标签



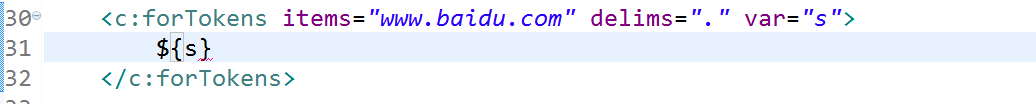
　　　　　　<c:forTokens > 标签

　　　　　　　　　　　　将自定义字符串，按照指定字符进行分割，并遍历输出

　　　　　　　　　　　　items：自定义字符串

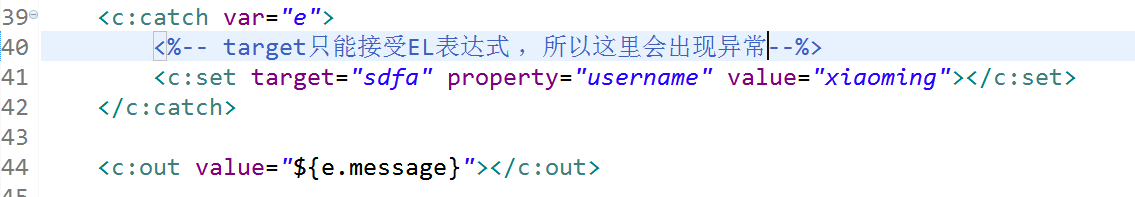
　　　　　　　　　　　　delims：指定字符

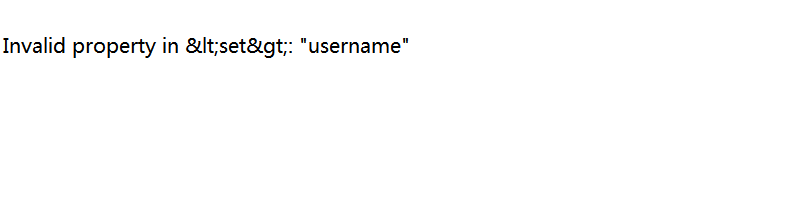
　　　　　　　　　　　　var：遍历的变量名。



　　　　　　<c:catch > 标签

　　　　　　　　　　　　相当于try catch





　　　　　　　　　　　　var：捕捉后的异常对象

　　　　　　　<c:forEach/>标签

　　　　　　　　　　增强的for循环，

　　　　　　　　　　格式：

　　　　　　　　　　　　<c:forEach items="" var="" />

　　　　　　　　　　　　items:需要遍历的对象

　　　　　　　　　　　　var:用于存放遍历的每一项内容，存放在page作用域，只能在循环体中使用

　　　　　　　　　　1、items为字符串或字符串数组

　　　　　　　　　　　　　　为字符串，直接输出，为字符串数组，遍历输出

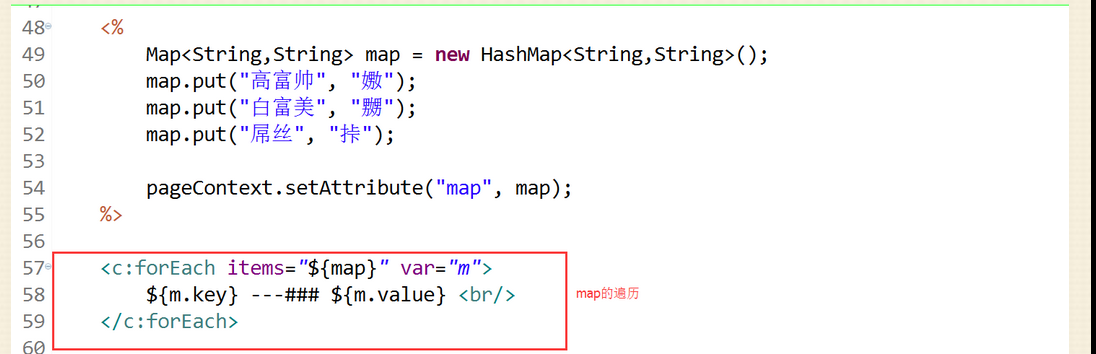


IMG_311

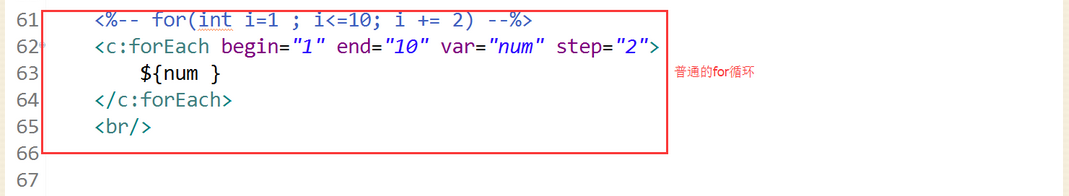
　　　　　　　　　　2、items为list集合



　　　　　　　　　　3、items为map集合



　　　　　　　　　　4、普通for循环



　　　　　　　　　　　　　　begin：从1开始

　　　　　　　　　　　　　　end：到10结束

　　　　　　　　　　　　　　step:每次加2

　　　　　　　　　　　　　　var：循环的变量，

　　　　　　<c:url  />标签

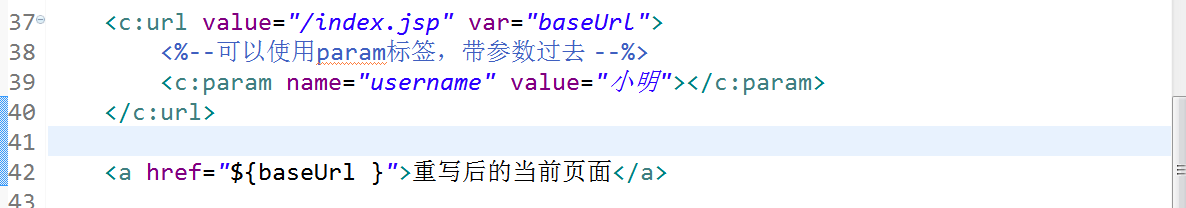
　　　　　　　　　　记得讲session的时候，如果cookie禁用了，那么只能将url进行重写才能让session继续生效，而使用的api是response.encodeURL();就是将SESSIONID当作参数带过来，而url标签的作用就是实现类似功能

　　　　　　　　　　<c:url value="" var="" scope="" />　　就讲解三个属性

　　　　　　　　　　　　value:要进行重写的url

　　　　　　　　　　　　var：如果写了该变量，那么就会将其重写后的url存入page作用域中，如果scope没有的话，默认就是page，否则以scope属性值为准，在作用域范围内就可以直接拿着这个变量使用了，如下面所展示的。

　　　　　　　　　　　　scope:作用域范围，默认是page，

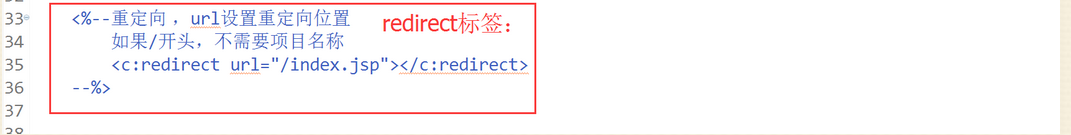


　　　　　　　　　　　　访问后点击超链接

　　　　　　　　　　　　　　没有关闭cookie，所以不会将SESSIONID写入url中 。

IMG_316

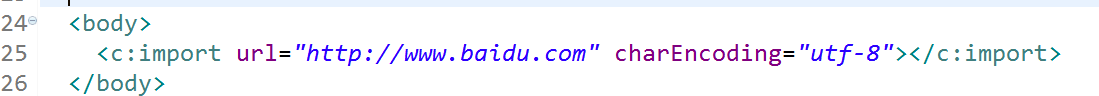
　　　　　　<c:redirect />标签



　　　　　　<c:import />标签

　　　　　　　　　　类似于JSP的include行为和include指令，JSTL也提供了include功能的标签<c:import/>，不过import标签功能更强大，甚至可以把Internet上的网页包含进来。

　　　　　　　　　　<c:import url="http://www.baidu.com" charEncoding="utf-8"/>





 　　　　fmt标签库

　　　　　　　　是一些辅助性功能标签，就讲解一个把，用的不多，等用到的时候在去详细学

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

　　　　　　　　标签只有一个属性，value

 　　　　　　　　<fmt:requestEncoding/>设置编码，

　　　　　　　　　　相当于Java中的request.setCharacterEncoding("");

　　　　　　　　　　post提交中文参数，使用<fmt:requestEncoding value="utf-8"/>来设置编码，就不会出现中文乱码了。

　　　　　　　　　　get提交中文参数，该标签就会失效，除非到TOMCAT的server.xml中，将URLEncoding设置为UTF-8(默认为ISO-8859-1)。

　　　　　　　　<fmt:setLocale/>显示所有地区的数据格式

　　　　　　　　等等

　　　　fn方法库

　　　　　　　　提供一些方法，函数，例如字符串查找，截取之类。之所以不叫fn标签库而叫fn方法库，因为与core，fmt标签格式不一样，并且必须在el表达式中使用

　　　　　　　　格式：fn:methodName()的格式，比如，fn:contains()

　　　　　　　　contains方法判断是否包含指定的字符串，相当于java里String类的contains方法。

　　　　　　　　　　　fn:contains(String string1，String string2);

　　　　　　　　containsIgnoreCase方法，判断是否包含指定字符串，忽略大小写

　　　　　　　　　　　fn:contains(String string1，String string2)

　　　　　　　　endsWith方法，判断某字符串是否以指定的参数结尾，

　　　　　　　　　　　fn:endWith(String string1，String string2);

　　　　　　　　startsWith方法，判断某字符串是否以指定参数开始

　　　　　　　　　　　fn:startsWith(String string1，String string2);

　　　　　　　　escapeXml方法，是否转义字符

　　　　　　　　　　　fn:escapeXml(source);

　　　　　　　　indexOf方法，查找字符串在另一个字符串中第一次出现的位置

　　　　　　　　　　　fn:indexOf(String string1，String string2);

　　　　　　　　split方法，将字符串以指定参数为界分割成多个字符串形成字符串数组

　　　　　　　　　　　。。

　　　　　　　　join方法，与split方法相反，将字符串数组以参数为界连接成字符串

　　　　　　　　length方法，取字符串，数组，集合的长度

　　　　　　　　等等

　　　　SQL标签库

　　　　　　　　JSP中直接可以操作数据库，但是大多数时候我们都不会这样用。。

　　　　XML标签库

　　　　　　　　为了方便地处理XML，

三、总结

　　　　重点讲解了一下JSTL中的core标签库，其实也就是一些控制流程的标签，没什么思想，只要会用就行了，以后忘记了就回过头来查查看，最重要的是思想，这些死东西记录一下就行了。