* JSP基本语法

**1.1. JSP的由来**

**1.1.1. 为什么有JSP规范**

Servlet技术产生以后，在使用过程中存在一个很大的问题，即为了表现页面的效果而需要输出大量的HTML标签，这些标签在Servlet中表现为一个个的字符串常量。这种输出页面的方式不仅仅增加了开发时对页面的控制难度，也不利于后期的维护。为了能够将Servlet中用于表现的功能分离出来，提高视图的开发效率，推出了JSP这种技术，主要用于将Servlet中负责显示的语句抽取出来。

**1.1.2. 什么是JSP**

JSP（Java Server Page）是Sun公司制定的一种服务器端动态页面技术的组件规范，以“.jsp”为后缀的文件中既包含HTML静态标记用于表现页面，也包含特殊的代码，用于生成动态内容。

JSP作为简化Servlet开发的一种技术，实质上最终依然要转变为Servlet才可能运行，只不过这个转变过程由Servlet容器来完成。所以遵循JSP的编写规范才能使得JSP转变为需要的Servlet。

**1.2. JSP编写规范**

**1.2.1. 如何编写JSP**

步骤一、创建一个以“jsp”为后缀的文件

步骤二、在文件中添加用于控制页面显示的HTML代码、样式及JavaScript脚本。

步骤三、在需要动态生成的部分添加Java代码的编程逻辑

**1.2.2. JSP页面中的HTML代码**

JSP页面中的HTML代码与静态的HTML页面代码没有区别，可以有基本标记，可以有用于控制样式的CSS，可以有控制页面的JS脚本。这些原本在Servlet中表现为字符串常量的内容最终依然会在JSP转换成Servlet时由系统使用out.write方法自动输出。

**1.2.3. JSP页面中的注释**

在JSP页面中可以添加如下两种类型的注释

01.<!-- 注释内容 -->

02.<%-- 注释内容 -->

第一种注释也叫HTML注释，可以出现在JSP页面之中，注释内容中可以包含了一些Java代码，但这些代码会被执行

第二行注释是JSP注释，不允许注释的内容出现Java代码，写了Java代码也会被忽略，不会执行。

**1.2.4. JSP页面中的Java代码**

JSP页面之中可以嵌入一些Java代码，以此动态生成页面中的一些内容。而JSP规范恰恰是针对这些嵌入的Java代码的一些约束，各种不同类型的代码的作用以及在转变为Servlet一部分时的规则都是不一样的。JSP页面中可以包含如下三种类型的Java代码：

•JSP表达式（方便输出）

•JSP小脚本（完成相对较长的逻辑运算）

•JSP声明（添加属性或方法）

这些Java代码可以在页面的任何位置进行编写，当他们被执行时就可以控制页面上产生可变化的内容了。

**1.2.5. JSP表达式**

使用表达式可以方便的在JSP页面中输出运算的结果，代码形式如下所示：

01.<%=3+5%>

02.<%=add()%>

03.<%=xx.getName()%>

04.<%=xx.getName()+“abc”%>

注意：表达式结束不需要写分号。

这种形式的Java代码在转译成Servlet时，会成为service（）方法中使用out.print语句的输出。

01.out.print(3+5);

02.out.print(add());

03.out.print(xx.getName());

04.out.print(xx.getName()+“abc”));

**1.2.6. JSP小脚本**

JSP小脚本可以编写Java代码段，从而实现相对较长的一段运算逻辑。这些Java代码最终都会成为Servlet中Service方法的一部分。由于HTML与Java可以进行混合使用，所以需要注意括号的匹配。

01.<table>

02.<%

03.List<User> allUser = (List<User>)request.getAttribute(“users“);

04.for(User u : allUser){

05.%>

06. <tr>

07. <td> <%=u.getId()%> </td>

08. <td> <%=u.getName()%> </td>

09. </tr>

10.<%

11.}

12.%>

13.</table>

以上代码使用小脚本控制生成一个表格。<tr>行标记作为要循环输出的一部分出现在了循环体内。需要注意循环体结束时的大括号的匹配。这些代码最终都会成为service方法的一部分，转换结果如下：

01.public void service(…){

02.out.write(“<table>”);

03.List<User> allUser = (List<User>)request.getAttribute(“users“);

04.for(User u : allUser){

05.out.write(“<tr> <td>”);

06.out.print(u.getId());

07.out.write(“</td><td>”);

08.out.print(u.getName());

09.out.write(“</td></tr>”);

10. }

11. out.write(“</table>”);

12.}

**1.2.7. JSP声明**

JSP声明可以为对应的Servlet添加属性和方法。这种形式的代码使用的很少。语法规则如下：

01.<%!

02. //属性或方法的声明

03.%>

如编写下列代码

01.<%!

02. public void fun(){

03. //… 方法体

04.}

05.%>

转换为Servlet之后的结果如下：

01.public class XXX\_JSP extends JSPBase{

02.public void fun(){

03. // … 方法体

05. }

06. public void service(… …){

08.}

09.}

**1.2.8. JSP页面中的指令**

指令在JSP页面中通常起到转译成Servlet时的格式控制的作用。基本语法为：

01.<%@ 指令名 属性=值 %>

常用指令包含以下三种

•page指令

•include指令

•taglib指令

这些指令都有自己的属性来实现不同的控制功能。taglib指令会在讲解JSP标签时详细介绍。

**1.2.9. page指令**

page指令可以实现在JSP页面中导入要用到的Java包，也可以对页面的一些属性进行设置。

导包

01.<%-- 导包 --%>

02.<%@ page import=“java.util.\*“%>

03.<%@ page import=“java.util.\*,java.sql.\*“%>

使用page指令导包时，需要用到import属性。如果需要导入多个包，可以分成多条page指令来编写，也可以在一条page指令中，使用“，“逗号作为分隔来实现。注意，page指令要放在页面的最上面编写

设置response.setContentType（）方法的参数值

01.<%-- 设置response.setConentType方法的参数值 --%>

02.<%@ page contentType=“text/html;charset=utf-8“%>

使用page指令可以设置输出内容的编码方式，这样就可以设置浏览器使用正确的解码方式来显示页面。

设置容器读取该文件时的解码方法

01.<%-- 设置容器读取该文件时的解码方式 --%>

02.<%@ page pageEncoding=“UTF-8“%>

为了保证页面中编写的中文能够保存，以及容器在加载文件时能正确的解码文件中的中文，需要通过page指令的pageEncoding属性来完成。这段代码可以保证页面在加载到内存时正确的解码中文。

**1.2.10. include指令**

include指令主要用于将其他页面包含在另一个页面之中。同一个应用中的很多页面都会有相同的导航头、版权声明，在每一个页面中重复编写是没有必要的，而且如果导航头发生变化时不利于所有页面的更新。如果将这样共用的部分单独做到一个文件中，只需要在不同的页面中添加引用，就可以实现页面内容的重用。

include指令的语法如下：

01.<%@ include file=“header.html” %>

**1.2.11. JSP页面中的隐含对象**

由于JSP页面最终要转化为一个Servlet，而作为即将转变为Servlet的一个前期版本，会有很多系统提供给页面能够直接使用的对象，这些对象的创建由系统完成，对象名也有系统命名，根据页面中要实现的功能直接使用即可。

**2. JSP运行原理**

**2.1. JSP的运行原理**

**2.1.1. JSP是如何运行的**

整个过程在Servlet容器的控制下将请求的JSP资源转变为Servlet之后，同样会经历Servlet的例化、初始化、就绪、销毁四部分的生命周期。第一次请求一个JSP页面会比之后的请求响应慢一些，就是要经过转译成Servlet，编译成class这样一个过程。

**2.1.2. JSP如何转换为Java**

JSP页面最终是要转译为一个Servlet，所以JSP页面中的HTML代码及Java代码、指令等内容都会按照固定的方式变为Servlet中的一部分。

用于导包的page指令最终成为import语句，page指令中的contentType属性转变为了\_JSPService()方法中的response.setContentType()方法，HTML标记都会有out.write方法输出，<%%>的小脚本原封不动成为了\_JSPService()方法内的一部分。当HTMl标记与小脚本混合在一起时，转变过程遵循从上至下的顺序逐行转变。

**2.1.3. 如何将静态页面转换为动态页面**

在将一个静态HTML页面转变为动态的JSP页面时，首先需要分析页面的整体结构，找到页面中不变的公共部分，如导航、页脚等信息。将静态页面的脚本拷贝到JSP页面之后，一定要添加page指令pageEncoding属性，保证页面中的中文能够被正确编码。添加page指令的import属性导入页面中需要的Java包。将页面中与目标页面不一致的地方进行修改，如修改表头与实际字段一致。最后将页面中需要动态生成的内容删除，使用小脚本的Java代码来实现运算逻辑。

* JSP开发要点

**1.1. 转发**

**1.1.1. 什么是转发**

在Web服务器端处理用户请求的时候，会有需要多个Web组件配合才能完成的情况。一个Web组件（Servlet/JSP）将未完成的处理通过容器转交给另外一个Web组件继续完成，这个转交的过程叫做转发。

常见情况是Servlet负责获取数据，然后将数据转交给JSP进行展现。

**1.1.2. 如何实现转发**

实现转发过程可遵循以下三个步骤：

步骤一、绑定数据到request对象

在转交的过程中一定会有数据的传递，并且涉及到的Web组件都是针对同一个请求，所以利用request来保存共同处理的数据不仅仅能让数据共享，也能够随着请求响应的结束而销毁，不会占用服务器更多的资源。使用如下代码可以实现数据的绑定:

01.req.setAttribute(String name,Object obj);

setAttribute()方法实现数据绑定，与其对应的还有getAttribute（）方法获取绑定的数据，removeAttribute（）方法移除绑定的数据。

步骤二、获得转发器

使用如下代码可以获取到转发器，用于说明转交的下一个组件的路径:

01.RequestDispatcher rd = req.getRequestDispatcher(String uri);

步骤三、实现转发

使用转发器完成转发的动作，因下一个Web组件要针对同一个请求和响应继续完成后续的工作，所以在转发时要将本次的请求和响应对象作为参数传给下一个Web组件。实现代码如下所示：

01.rd.forward(req,res);

其中步骤二和步骤三可以合并为一行代码：

01.request.getRequestDispatcher(String uri).forward(request,response);

**1.1.3. 转发的原理**

(1.请求到达服务器(2.根据请求信息创建request和response对象(3.根据请求资源路径找到对应的Servlet执行处理(4.Servlet在处理过程中从数据库获取到结果信息(5.Servlet将结果信息绑定在request对象中(6.Servlet通知容器将request和response对象转交给list.jsp页面继续执行对请求的响应(7. list.jsp页面被调用并执行时从传递过来的request对象中获取绑定的数据生成结果页面(8.服务器将list.jsp页面的执行结果返回给客户端。

**1.1.4. 转发的特点**

转发过程发生在服务器端，客户端只发送了一个请求，虽然请求到达服务器的指定位置后被容器控制着传到了第二个组件继续完成工作，但浏览器并不知道这个过程，所以转发之后地址栏地址不会发生变化。

转发的目的地必须是同一个应用内部的某个地址，决不能跳出应用。毕竟这个转交过程由容器实现，容器只有访问本应用的权限，而不能控制请求到达应用以外的位置。

转发过程中涉及到的所有Web组件共享同一个request对象和response对象，数据的传递和共享就依赖request对象。

注意：在forward之后的其他语句还是会继续执行完的，只要不报异常。

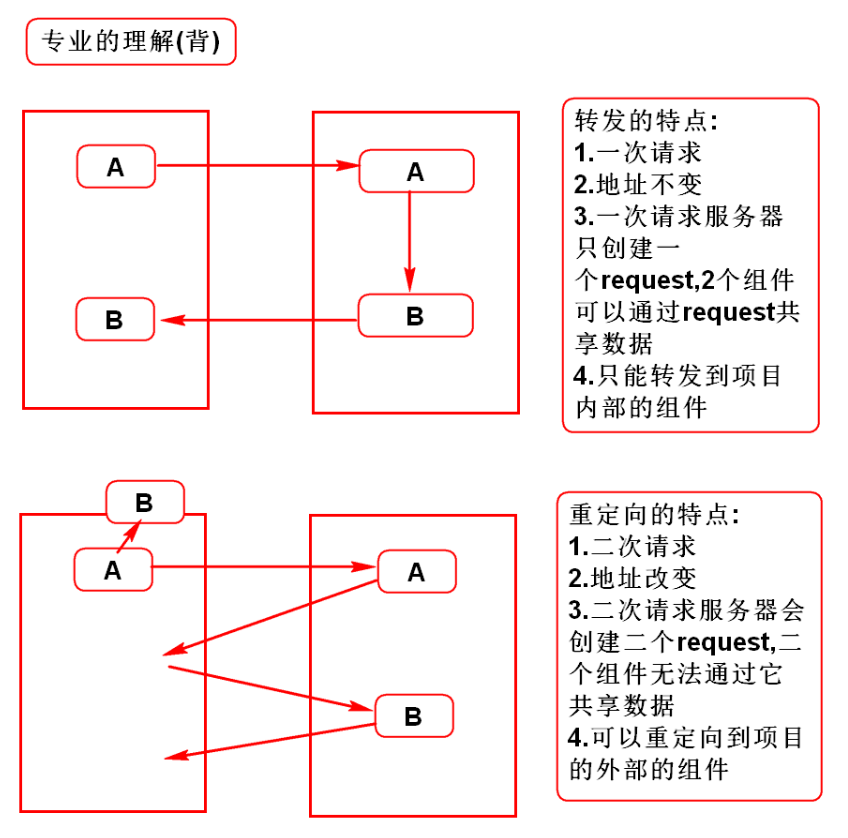
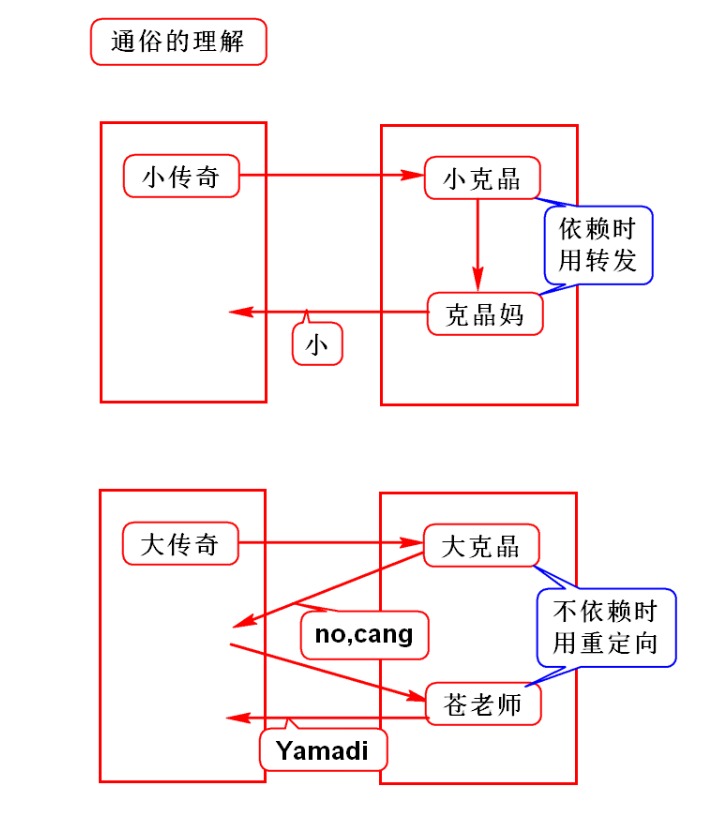
**1.1.5. 转发和重定向的区别**

重定向：浏览器发送请求到容器访问A，A可以发送一个状态码302和一个Location消息头到浏览器，于是浏览器会立即向Location发新的请求。

转发：浏览器发送请求到容器访问A，A可以通知容器去调用B。转发所涉及的各个Web组件会共享同一个request和response对象；而重定向不行。

说明：当请求到达容器，容器会创建request对象和response对象。当响应发送完毕，容器会立即删除request对象和response对象。即，request对象和response对象的生存时间是一次请求与响应期间。

转发之后，浏览器地址栏的地址不变，重定向会变。转发的地址必须是同一个应用内部某个地址，而重定向没有这个限制。转发是一件事情未做完，调用另外一个组件继续做；而重定向是一件事情已经做完，再做另外一件事情。



**1.2. 异常处理**

**1.2.1. 编程式异常处理**

在编写Web类型的应用时，如果程序运行发生了异常就会返回该异常的信息到客户端的浏览器上显示出来，这些信息往往暴露了服务器的一些重要结构，所以要避免异常信息输出到客户端。一般有两种方式来处理异常，编程式的异常处理和声明式的处理。

编程式的异常处理就是在程序中捕获到异常时，使用转发跳转到指定页面进行提示说明。

09. try {

10. // 使用dao访问数据库

11. //交给jsp来完成

12. request.getRequestDispatcher("list3.jsp").forward(request,

13.response);

14. } catch (Exception e) {

15. e.printStackTrace();

16. //使用转发的方式来处理异常。

17.request.setAttribute("error\_msg","系统繁忙，稍后重试");

18. request.getRequestDispatcher("error.jsp").forward(request,

19. response);

**1.2.2. 容器中声明式处理**

声明式的处理主要依靠容器自己来完成，即产生异常时抛出给容器，但不能让容器将这些底层信息返回给客户端，所以需要在web.xml文件中添加配置说明，通知容器在捕获到异常时应该将什么样的页面返回给客户端。一旦使用声明式处理方式，则该项目下的任意一个文件错误或异常，都会跳到指定的错误处理页面。

具体的实现步骤如下：

步骤一、在代码中捕获到异常直接抛出（注意异常类型，必须是ServletException）

代码结构如下图所示：

try {

10. // 使用dao访问数据库

11. //交给jsp来完成

12. request.getRequestDispatcher("list3.jsp").forward(request,

13.response);

14. } catch (Exception e) {

15. e.printStackTrace();

16. //将异常抛给容器来处理

27. <servlet>

28. <servlet-name>ActionServlet</servlet-name>

29. <servlet-class>web.ActionServlet</servlet-class>

30. </servlet>

31. <servlet-mapping>

32. <servlet-name>ActionServlet</servlet-name>

33. <url-pattern>\*.do</url-pattern>

34. </servlet-mapping>

35. <!-- 配置错误处理页面 -->

36.<error-page>

37. <exception-type>javax.servlet.ServletException</exception-type>

38. <location>/error.jsp</location>

39. </error-page>

40.</web-app>

**1.2.3. 不同异常处理方式的应用场景**

程序级的异常最好选择编程式的处理方法，利用转发到达指定的错误处理页面。

系统级别的异常最好选择声明式的处理方法。

**1.3. 路径问题**

**1.3.1. 什么是路径**

在JSP页面或Servlet中需要从一个组件到另一个组件的跳转，通常以链接、表单提交、重定向、转发的形式来完成，其中对目标位置的标识即路径。

•链接地址<a href=”url”>链接文本</a>

•表单提交<form action=”url”>

•重定向response.sendRedirect（url）；

•转发request.getRequestDispatcher（url）；

**1.3.2. 什么是相对路径**

相对路径指的是相对于当前位置，为了到达目标文件要经过的路径。书写格式上不以“/“开头，如果为了退至上一级目录可以”../“开头。

从index.jsp访问a2.jsp，使用相对路径，具体值为“a/a2.jsp“。

从a1.jsp访问index.jsp，使用相对路径，具体值为“../index.jsp“。

**1.3.3. 什么是绝对路径**

绝对路径指的是，不管当前文档所在的位置在应用的哪里，都会从一个固定的点出发，然后构建到达目标文件所需要经过的路径。通常绝对路径在书写格式上以“/”开头。这个固定的点可能是应用名，也可能是应用名之后。

从index.jsp访问a2.jsp，使用绝对路径，非转发的情况下具体值为“/appName/a/a2.jsp“。

从a1.jsp访问index.jsp，使用绝对路径，非转发的情况下具体值为“/appName/index.jsp“。

**1.3.4. 路径的处理技巧**

对于以下四种常用的需要写路径的代码，在编写绝对路径时起始点是不同的，具体区别如下：

•链接地址<a href=”/appName”>链接文本</a>

•表单提交<form action=”/appName”>

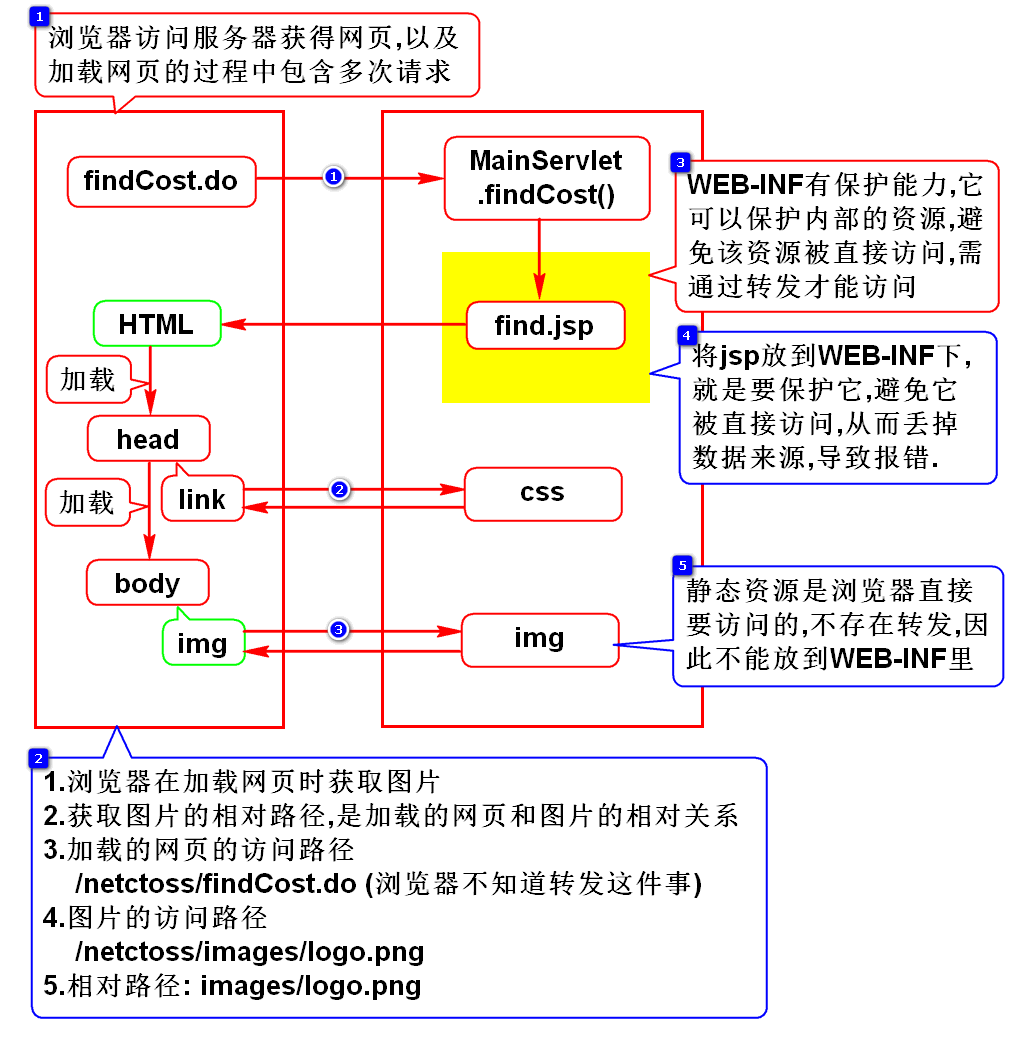
•重定向response.sendRedirect（”/appName”）；

•转发request.getRequestDispatcher（”/xxx”）；

链接地址，表单提交，重定向由于都是浏览器发出的请求，为了到达指定的应用内的资源，所以斜杠后面从应用名开始写；转发位于服务器端，已经在具体的应用内部了，所以斜杠后面从应用名之后开始写。

由于Web应用的真实部署名称和测试时的名称未必一样，所以使用绝对路径时如果应用名写成定值则会带来变更的麻烦。为了获取当前应用的实际部署名称可以使用如下代码来动态获取。

1. String request.getContextPath();



* EL表达式

**1.1. EL表达式**

**1.1.1. 为什么需要EL表达式和JSP标签**

JSP的产生在一定程度上将Servlet中负责表现的功能抽取了出来，但JSP页内嵌入的Java代码也破坏了页面中负责表现的页面结构，特别是当运算逻辑稍微复杂一点的话，那么JSP页面中大量的Java代码增加了页面维护的难度。所以使用简单的标签来表现复杂**的逻辑以及使用简单的形式表现运算的关系就是EL表达式和JSP标签出现的原因。**

**1.1.2. 什么是EL表达式**

一套简单的运算规则，用于给JSTL标签的属性赋值，也可以直接用来输出而脱离标签单独使用。

**1.1.3. EL表达式的作用**

EL（Expression Language）是从 JavaScript 脚本语言得到启发的一种表达式语言，它借鉴了 JavaScript 多类型转换无关性的特点。在使用 EL 从 scope 中得到参数时可以自动转换类型，因此对于类型的限制更加宽松。

**1.1.4. 访问Bean属性**

在JSP页面中经常要输出系统定义的对象的属性，而按照以往的写法需要自己去对象域中获取、转换再输出，使用EL表达式可以非常明显的简化过程。

Bean：指的是一个公共的类，按照固定的方式提供属性的get/set访问方式。针对这种特殊类型的属性访问使用EL表达式实现有两种方式，如下。

**1.1.5. 方式一：${对象名.属性名}**

${user.name}

执行的过程为：从pageContext、request、session、application中依次查找绑定名为“user”的对象，找到后调用“getName”方法，将返回值输出。

假定在session中绑定了一个对象，如下：

01.User obj = new User(1，“胡萝卜”)；

02.session.setAttribute(“user”，obj)；

那么${user.name}等价于下面代码：

01.<%

02. User u = (User)session.getAttribute(“user”);

03. out.print(u.getName();

04.%>

这种繁琐的取值，转换，输出的过程就都由系统代劳了。而且表达式比以上繁琐代码更会处理null。如果没有为name属性赋过值，页面输出“”，不会输出null。如果取值时绑定名写错，如${obj.name}，页面也会输出“”，而不是报空指针异常。但属性名写错会报错，如${user.naaa}.

**1.1.6. 方式二：${对象名[“属性名”]}**

表达式也支持属性名的动态读取，这时需要采用方式二${user[“name”]}的形式。

假定在Servlet中有如下代码：

01.User obj = new User(1,”胡萝卜”);

02.session.setAttribute(“user”,obj);

03.session.setAttribute(“pName”，”id”);

在JSP中编写如下代码会输出“1”:

01.${sessionScope.user[“id“]}

在JSP中编写如下代码也会输出“1”:

01.${sessionScope.user[sessionScope.pName]}

如果pName在绑定时不指定id，而是name，那么这个表达式就会输出“胡萝卜“，所以这种写法可以认为是表达式中有一个变量。sessionScope.pName 等价于 session.getAttribute(“pName”)。

那么对于interests这个数组属性的值可以使用如下表达式访问：

01.${user.interests[0]}

**1.1.7. 指定对象的查找范围**

在书写表达式时，如果没有指定搜索范围，那么系统会依次调用pageContext、request、session、application的getAttribute（）方法。这样不限定查找范围的代码不利于排错，所以这种取值的操作可以限定对象的查找范围。如：

01.${sessionScope.user.name}

一旦指定了对象所在的范围，那么只会在范围内查找绑定对象，不会在找不到的时候再去其他区域中查找了。

sessionScope的位置还可以填写pageScope、requestScope、applicationScope。

**1.1.8. 使用EL表达式进行计算**

使用EL表达式可以单独进行运算得出结果并直接输出，如下代码所示，EL进行算术运算，逻辑运算，关系运算，及empty运算。空运算主要用于判断字符串，集合是否为空，是空或为null及找不到值时都会输出true。

01.<%request.getSession().setAttribute("sampleValue", new Integer(10));%>

03.${sessionScope.sampleValue} // 显示 10

04.${sessionScope.sampleValue + 12} <br> // 显示22

05.${(sessionScope.sampleValue + 12)/3} <br> // 显示7.3

06.${(sessionScope.sampleValue + 12) /3==4} <br> // 显示 false

07.${(sessionScope.sampleValue + 12) /3>=5} <br> // 显示 true

08.<input type="text" name="sample1" value="${sessionScope.sampleValue + 10}">

09.// 显示值为20的 Text 控件

10.${empty null}//显示true

**1.1.9. 使用EL表达式获取请求参数值**

以下两种写法分别等价：

01.${param.username} 与 request.getParameter(“username”);

02.${paramValues.city} 与request.getParameterValues("city")；

* JSTL

**2.1. JSTL**

**2.1.1. 什么是JSTL**

Sun 公司 Java 标准规范的 JSTL 由 apache组织负责维护。作为开源的标准技术，它一直在不断地完善。 JSTL 的发布包有两个版本： Standard-1.0 Taglib 、 Standard-1.1 Taglib ，它们在使用时是不同的。

**2.1.2. 如何使用JSTL**

将标签库对应的jar包拷贝到WEB-INF/lib目录下，以便于系统可以加载所需要的类。使用taglib指令在页面上引入标签的命名空间和前缀，帮助系统定位对应的类。

如：

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

**2.1.3. 核心标签-if标签**

使用if标签实现属性判断后的输出。代码如下所示：

01.<%

02. User user = new User();

03. user.setName("胡萝卜");

04. user.setGender("f");

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

06.%>

07. 姓名:${user.name}<br/>

08. 性别:

09.<c:if test="${user.gender =='m'}" var="rs" scope="request">男</c:if>

10.<c:if test="${!rs}">女</c:if>

**2.1.4. 核心标签-choose标签**

使用choose标签简化多个if标签的判断。代码如下所示：

01.<%

02. User user = new User();

03. user.setName("胡萝卜");

04. user.setGender("x");

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

06.%>

07. 性别:

08.<c:choose>

09. <c:when test="${user.gender == 'm'}">男</c:when>

10. <c:when test="${user.gender =='f'}">女</c:when>

11. <c:otherwise>未知</c:otherwise>

12.</c:choose>

**2.1.5. 核心标签-forEach标签**

使用forEach标签完成对集合的遍历输出。

其中items属性为要遍历的集合，var属性为每次取出来的一个对象，varStatus指定当前迭代的状态。代码如下：

01.<table>

02. <tr>

03. <td>序号</td>

04. <td>姓名</td>

05. <td>年龄</td>

06. </tr>

07. <c:forEach items="${users}" var="u" varStatus="s">

08. <tr>

09. <td>${s.count}</td>

10. <td>${u.name}</td>

11. <td>${u.age}</td>

12. </tr>

13. </c:forEach>

14.</table>

**2.1.6. 如何开发自定义标签**

编写一个继承自SimpleTagSupport的Java类：

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

02.public class HelloTag extends SimpleTagSupport{

04.}

重写该类的doTag方法，在其中添加处理逻辑：

17. public void doTag() throws JspException, IOException {

18. PageContext ctx =(PageContext)getJspContext();

19. JspWriter out = ctx.getOut();

20. for(int i=0;i< qty;i++){

21. out.println(info+"<br/>");

在WEB-INF下面新建一个tld文件，用于配置标签说明文件。代码如下

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

02.<taglib xmlns="http://java.sun.com/xml/ns/j2ee"

03. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

04. xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary\_2\_0.xsd" version="2.0">

06.<tlib-version>1.1</tlib-version>

07.<short-name>c1</short-name>

08.<uri>http://www.tarena.com.cn/mytag</uri>

09.<tag>

10. <name>hello</name>

11. <tag-class>tag.HelloTag</tag-class>

12. <body-content>empty</body-content>

13. <attribute>

14. <name>info</name>

15. <required>true</required>

16. <rtexprvalue>false</rtexprvalue>

17. </attribute>

18. <attribute>

19. <name>qty</name>

20. <required>true</required>

21. <rtexprvalue>true</rtexprvalue>

22. </attribute>

23.</tag>

24.</taglib>

标签在页面中被引用时的代码如下：

01.<%@taglib uri="http://www.tarena.com.cn/mytag" prefix="c1" %

02.//… …

03.<c1:hello info="hello kitty" qty="${1+5}"/>

**2.1.7. 标签的运行原理**

容器依据JSP页面中的uri找到tld文件(依据标签中的<c1:hello>hello这个名字找到标签类tag.HelloTag。接下来实例化该标签，同时属性值赋给参数，调用doTag方法。

