[**EL表达式详解**](http://www.blogjava.net/supercrsky/articles/179363.html)

在 JSP 页面中，使用标签库代替传统的 Java 片段语言来实现页面的显示逻辑已经不是新技术了，然而，由自定义标签很容易造成重复定义和非标准的实现。鉴于此，出现了 JSTL （ JSP Standard Tag Library ）。大多数 JSP 页面逻辑提供了实现的 JSTL 技术，该技术本身就是一个标签库。

Sun 公司 Java 规范标准的 JSTL 由 apache jakarta 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。

         Standard-1.0 Taglib （ JSTL1.0 ）支持 Servlet2.3 和 JSP1.2 规范， Web 应用服务器 Tomcat4 支持这些规范，而它的发布也在 Tomcat 4.1.24测试通过了。

         Standard-1.1 Taglib （ JSTL1.1 ）支持 Servlet2.4 和 JSP2.0 规范， Web 应用服务器 Tomcat5 支持这些规范，它的发布在 Tomcat 5.0.3 测试通过了。

在本章的介绍中，将以由 Sun 发布的 Standard-1.1 Taglib 标签库为主，而 apache jakarta 组织发布的开源标签库，可以从 http://jakarta.apache.org/taglibs/ 找到所需要的帮助。 Sun 发布的标准 JSTL1.1 标签库有以下几个标签：

         核心标签库：包含 Web 应用的常见工作，比如：循环、表达式赋值、基本输入输出等。

         国际化标签库：用来格式化显示数据的工作，比如：对不同区域的日期格式化等。

         数据库标签库：可以做访问数据库的工作。

         XML 标签库：用来访问 XML 文件的工作，这是 JSTL 标签库的一个特点。

         函数标签库：用来读取已经定义的某个函数。

此外， JSTL 还提供了 EL 表达式语言（ Expression Language ）来进行辅助的工作。

**JSTL 标签 库由标签库和 EL 表达式语言两个部分组成。** EL 在 JSTL 1.0 规范中被引入，当时用来作为 Java 表达式来工作，而该表达式必须配合 JSTL的标签库才能得到需要的结果。

说明：在 JSTL 1.1 规范中， JSP2.0 容器已经能够独立的理解任何 EL 表达式。 EL 可以独立出现在JSP 页面的任何角落。本文随后的内容将以 JSTL 1.1 规范作为介绍的重点。

**一 JSTL EL表达式语言简介**

EL 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。 Web 服务器对于 request 请求参数通常会以 String 类型来发送，在得到时使用的 Java 语言脚本就应该是request.getParameter(“XXX”) ，这样的话，对于实际应用还必须进行强制类型转换。而 EL 就将用户从这种类型转换的繁琐工作脱离出来，允许用户直接使用EL 表达式取得的值，而不用关心它是什么类型。

下面的示例就是一个 EL 表达式，见例 1 。

例 1 ：简单 EL 表达式

http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockStart.gif<%@ page contentType="text/html; charset=UTF-8"%>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<html>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif <body>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    ${sampleValue + 1} <br>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif </body>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</html>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif

这个示例将在 JSP 页面显示为“ 1 ”， EL 表达式必须以“ ${XXX} ”来表示，其中“ XXX ”部分就是具体表达式内容，“ ${} ”将这个表达式内容包含在其中作为 EL 表达式的定义。本示例可以在满足 JSP2.0 规范的任何 Web 应用服务器中使用。

**二 EL表达式的默认变量**

一个 EL 表达式包含变量和操作符两个内容。任何存在于 JSP 作用范围的 JavaBean 都可以被转化成 EL 表达式来使用，它所包含的默认变量如下：

**1 ．默认变量 pageScope 、 requestScope 、 sessionScope 、 applicationScope**

这 4 个默认变量包含**Scope作用范围的参数**集合，相当于被保存在 java.util.Map 中的某个参数。下面看简单的示例 2 ：

例 2 ：使用 sessionScope 变量的 EL 表达式

http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockStart.gif<%request.getSession().setAttribute("sampleValue", new Integer(10));%>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif${sessionScope.sampleValue}   
http://www.blogjava.net/Images/OutliningIndicators/None.gif

取得保存在 Session 中参数的 sessionScope 变量的 EL 表达式，“ . ”是 property 访问操作符，在这里表示从 Session 中取得“键”为“sampleValue”的参数，并显示出来。显示结果为“ 10 ”。

**2 ．默认变量 param 、 paramValues**

这两个默认变量包含**请求参数**的集合，param 表明请求包含的参数为单一控件， paramValues 表明请求包含的参数为控件数组。下面看一个简单示例 3：

例 3 ：提交请求的页面和接受的页面

http://www.blogjava.net/Images/OutliningIndicators/ExpandedBlockStart.gif<%@ page contentType="text/html; charset=UTF-8"%>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<html>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif <body>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif      <form action="SampleJsp.jsp">   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <input type="text" name="sampleValue" value="10">   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <input type="text" name="sampleValue" value="11">   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <input type="text" name="sampleValue" value="12">   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <input type="text" name="sampleSingleValue" value="SingleValue">   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    <input type="submit" value="Submit">   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    </form>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif </body>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</html>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif

在这个页面中定义了两组控件，控件名为“ sampleValue ”的是一套控件数组，控件名为“ sampleSingleValue ”的是单一控件，通过递交将请求参数传送到 SampleJsp.jsp 。

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

http://www.blogjava.net/Images/OutliningIndicators/None.gif<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<html>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif <body>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    ${paramValues.sampleValue[2]} <br>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif    ${param.sampleSingleValue} <br>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif </body>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif</html>   
http://www.blogjava.net/Images/OutliningIndicators/None.gif

这是请求转发到的页面，通过 EL 表达式的 paramValues 变量得到控件数组中最后一个控件的递交参数，通过 EL 表达式的 param 变量得到单一控件的递交参数。控件数组参数的 EL 表达式使用“ [] ”来指定数组下标。本示例将显示控件数组中最后一个控件的值“ 12 ”和单一控件的值“ SingleValue ”。

**3 ．默认变量 header 、 headerValues**

这两个默认变量包含**请求参数头部信息**的集合， header 变量表示单一头部信息， headerValues 则表示数组型的头部信息。

**4 ．默认变量 cookie**

包含**所有请求的cookie**集合，集合中的每个对象对应 javax.servlet.http.Cookie 。

**5 ．默认变量 initParam**

包含**所有应用程序初始化参数**的集合。

**6 ．默认变量 pageContext**

等价于 page 环境类 javax.servlet.jsp.PageContext 的实例，用来**提供访问不同的请求参数**。

pageContext详解

我们可以使用 ${pageContext}来取得其他有关用户要求或页面的详细信息。下表列出了几个比较常用的部分

|  |  |
| --- | --- |
| Expression | 说明 |
| ${pageContext.request.queryString} | 取得请求的参数字符串 |
| ${pageContext.request.requestURL} | 取得请求的URL，但不包括请求之参数字符串,即servlet的HTTP地址。 |
| ${pageContext.request.contextPath} | 服务的webapplication的名称 |
| ${pageContext.request.method} | 取得HTTP的方法(GET、POST) |
| ${pageContext.request.protocol} | 取得使用的协议(HTTP/1.1、HTTP/1.0) |
| ${pageContext.request.remoteUser} | 取得用户名称 |
| ${pageContext.request.remoteAddr} | 取得用户的IP地址 |
| ${pageContext.session.new} | 判断session是否为新的，所谓新的session，表示刚由server产生而client尚未使用 |
| ${pageContext.session.id} | 取得session的ID |
| ${pageContext.servletContext.serverInfo} | 取得主机端的服务信息 |

 这个对象可有效地改善代码的硬编码问题，如页面中有一A标签链接访问一个SERVLET，如果写死了该SERVLET的HTTP地址那么如果当该SERVLET的SERVLET-MAPPING改变的时候必须要修改源代码，这样维护性会大打折扣。

11 个默认变量几乎包含了 Web 应用的所有基本操作，若一个表达式不使用这些变量而直接使用参数名，那么就采用**就近原则**。该表达式将使用最近取得的参数值。

**EL 隐含对象**

JSP有9个隐含对象，而EL也有自己的隐含对象。EL隐含对象总共有11 个

|  |  |  |
| --- | --- | --- |
| **隐含对象** | **类型** | **说明** |
| **PageContext** | **javax.servlet.ServletContext** | **表示此JSP的PageContext** |
| **PageScope** | **java.util.Map** | **取得Page范围的属性名称所对应的值** |
| **RequestScope** | **java.util.Map** | **取得Request范围的属性名称所对应的值** |
| **sessionScope** | **java.util.Map** | **取得Session范围的属性名称所对应的值** |
| **applicationScope** | **java.util.Map** | **取得Application范围的属性名称所对应的值** |
| **param** | **java.util.Map** | **如同ServletRequest.getParameter(String name)。回传String类型的值** |
| **paramValues** | **java.util.Map** | **如同ServletRequest.getParameterValues(String name)。回传String[]类型的值** |
| **header** | **java.util.Map** | **如同ServletRequest.getHeader(String name)。回传String类型的值** |
| **headerValues** | **java.util.Map** | **如同ServletRequest.getHeaders(String name)。回传String[]类型的值** |
| **cookie** | **java.util.Map** | **如同HttpServletRequest.getCookies()** |
| **initParam** | **java.util.Map** | **如同ServletContext.getInitParameter(String name)。回传String类型的值** |

**三.EL表达式的操作符**

EL表达式中还有许多操作符可以帮助完成各种所需的操作，之前的示例中“.”、“[]”就是其中的两个，下面将用表9.1来展示所有操作符及它们各自的功能。

表 9.1 EL 表达式的操作符

|  |  |
| --- | --- |
| 操作符 | 功能和作用 |
| . | 访问一个 bean 属性或者 Map entry |
| [] | 访问一个数组或者链表元素 |
| () | 对子表达式分组，用来改变赋值顺序 |
| ? : | 条件语句，比如：条件 ?ifTrue:ifFalse  如果条件为真，表达式值为前者，反之为后者 |
| + | 数学运算符，加操作 |
| - | 数学运算符，减操作或者对一个值取反 |
| \* | 数学运算符，乘操作 |
| / 或 div | 数学运算符，除操作 |
| % 或mod | 数学运算符，模操作 ( 取余 ) |
| == 或 eq | 逻辑运算符，判断符号左右两端是否相等，如果相等返回 true ，否则返回 false |
| != 或 ne | 逻辑运算符，判断符号左右两端是否不相等，如果不相等返回 true ，否则返回false |
| < 或 lt | 逻辑运算符，判断符号左边是否小于右边，如果小于返回 true ，否则返回 false |
| > 或 gt | 逻辑运算符，判断符号左边是否大于右边，如果大于返回 true ，否则返回 false |
| <= 或 le | 逻辑运算符，判断符号左边是否小于或者等于右边，如果小于或者等于返回 true，否则返回 false |
| >= 或 ge | 逻辑运算符，判断符号左边是否大于或者等于右边，如果大于或者等于返回 true，否则返回 false |
| && 或and | 逻辑运算符，与操作赋。如果左右两边同为 true 返回 true ，否则返回 false |
| || 或 or | 逻辑运算符，或操作赋。如果左右两边有任何一边为 true 返回 true ，否则返回false |
| ! 或 not | 逻辑运算符，非操作赋。如果对 true 取运算返回 false ，否则返回 true |
| empty | 用来对一个空变量值进行判断 : null 、一个空 String 、空数组、 空 Map 、没有条目的 Collection 集合 |
| func(args) | 调用方法 , func 是方法名， args 是参数，可以没有，或者有一个、多个参数 . 参数间用逗号隔开 |

这些操作符都是极其有用的，下面通过几个示例来演示它们的使用方法：

例 4 ：几组操作符的示例

http://www.blogjava.net/Images/OutliningIndicators/None.gif${pageScope.sampleValue + 12} <br>           // 显示 12   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif${(pageScope.sampleValue + 12)/3} <br>     // 显示 4.0   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif${(pageScope.sampleValue + 12) /3==4} <br>         // 显示 true   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif${(pageScope.sampleValue + 12) /3>=5} <br>         // 显示 false   
http://www.blogjava.net/Images/OutliningIndicators/None.gif  
http://www.blogjava.net/Images/OutliningIndicators/None.gif<input type="text" name="sample1" value="${pageScope.sampleValue + 10}"> // 显示值为 10 的 Text 控件   
http://www.blogjava.net/Images/OutliningIndicators/None.gif

**EL逻辑运算符范例**

|  |  |  |
| --- | --- | --- |
| 逻辑运算符 | 范例 | 结果 |
| &&或and | 交集${A && B}或${A and B} | true/false |
| ||或or | 并集${A || B}或${A or B} | true/false |
| !或not | 非${! A }或${not A} | true/false |
| ==或eq | ${param.id == param.packagesId} | true/false |

**可以看到，对于这些示例，程序设计者完全无需管理它们的类型转换，在表达式内部都已经处理了(自动转换)。有了 EL 表达式，在 JSP 页面的编程变得更灵活，也更容易。大家应该很明确EL表达式只能通过内置对象取值，也就是只读操作，如果想进行写操作的话就让后台代码去完成，毕竟EL表达式仅仅是视图上的输出标签罢了。**