**EL 全名为Expression Language**

**EL 语法很简单，它最大的特点就是使用上很方便。接下来介绍EL主要的语法结构：**

**${sessionScope.user.sex}**

**所有EL都是以${为起始、以}为结尾的。上述EL范例的意思是：从Session的范围中，取得**

**用户的性别。假若依照之前JSP Scriptlet的写法如下：**

**User user = (User)session.getAttribute("user");**

**String sex = user.getSex( );**

**两者相比较之下，可以发现EL 的语法比传统JSP Scriptlet 更为方便、简洁。**

**.与 [ ] 运算符**

**EL 提供 . 和 [ ] 两种运算符来导航数据。下列两者所代表的意思是一样的：**

**${sessionScope.user.sex}等于${sessionScope.user["sex"]}**

**. 和 [ ] 也可以同时混合使用，如下：**

**${sessionScope.shoppingCart[0].price}**

**回传结果为shoppingCart中第一项物品的价格。**

**不过，以下两种情况，两者会有差异：**

**(1) 当要存取的属性名称中包含一些特殊字符，如. 或 – 等并非字母或数字的符号，就一定要使用 [ ]，例如：${user.My-Name }**

**上述是不正确的方式，应当改为：${user["My-Name"] }**

**(2) 我们来考虑下列情况：**

**${sessionScope.user[data]}**

**此时，data 是一个变量，假若data的值为"sex"时，那上述的例子等于${sessionScope.user.sex}；**

**假若data 的值为"name"时，它就等于${sessionScope.user.name}。因此，如果要动态取值时，就可以用上述的方法来做，但. 无法做到动态取值。**

**EL 变量**

**EL 存取变量数据的方法很简单，例如：${username}。它的意思是取出某一范围中名称为username的变量。因为我们并没有指定哪一个范围的username，所以它的默认值会先从Page 范围找，假如找不到，再依序到Request、Session、Application范围。假如途中找到username，就直接回传，不再继续找下去，但是假如全部的范围都没有找到时，就回传null，当然EL表达式还会做出优化，页面上显示空白，而不是打印输出NULL。**

|  |  |
| --- | --- |
| **属性范围（jstl名称）** | **EL中的名称** |
| **Page** | **PageScope** |
| **Request** | **RequestScope** |
| **Session** | **SessionScope** |
| **Application** | **ApplicationScope** |

**我们也可以指定要取出哪一个范围的变量：**

|  |  |
| --- | --- |
| **范例** | **说明** |
| **${pageScope.username}** | **取出Page范围的username变量** |
| **${requestScope.username}** | **取出Request范围的username变量** |
| **${sessionScope.username}** | **取出Session范围的username变量** |
| **${applicationScope.username}** | **取出Application范围的username变量** |

**其中，pageScope、requestScope、sessionScope和applicationScope都是EL 的隐含对象，由它们的名称可以很容易猜出它们所代表的意思，例如：${sessionScope.username}是取出Session范围的username 变量。这种写法是不是比之前JSP 的写法：**

**String username = (String) session.getAttribute("username");容易、简洁许多.**

**自动转变类型**

**EL 除了提供方便存取变量的语法之外，它另外一个方便的功能就是：自动转变类型，我们来看下面这个范例：**

**${param.count + 20}**

**假若窗体传来count的值为10时，那么上面的结果为30。之前在JSP 之中不能这样做，原因是从窗体所传来的值，它们的类型一律是String，所以当你接收之后，必须再将它转为其他类型，如：int、float 等等，然后才能执行一些数学运算，下面是之前的做法：**

**String str\_count = request.getParameter("count");**

**int count = Integer.parseInt(str\_count);**

**count = count + 20;**

**所以，注意不要和java的语法（当字符串和数字用“+”链接时会把数字转换为字符串）搞混淆喽。**

**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** |

**不过有一点要注意的是如果你要用EL输出一个常量的话，字符串要加双引号，不然的话EL会默认把你认为的常量当做一个变量来处理，这时如果这个变量在4个声明范围不存在的话会输出空，如果存在则输出该变量的值。**

**属性(Attribute)与范围(Scope)**

**与范围有关的EL 隐含对象包含以下四个：pageScope、requestScope、sessionScope 和**

**applicationScope，它们基本上就和JSP的pageContext、request、session和application一样，所以笔者在这里只稍略说明。不过必须注意的是，这四个隐含对象只能用来取得范围属性值，即JSP中的getAttribute(String name)，却不能取得其他相关信息，例如：JSP中的request对象除可以存取属性之外，还可以取得用户的请求参数或表头信息等等。但是在EL中，它就只能单纯用来取得对应范围的属性值，例如：我们要在session 中储存一个属性，它的名称为username，在JSP 中使用session.getAttribute("username") 来取得username 的值， 但是在EL中，则是使用${sessionScope.username}来取得其值的。**

**cookie**

**所谓的cookie是一个小小的文本文件，它是以key、value的方式将Session Tracking的内容记录在这个文本文件内，这个文本文件通常存在于浏览器的暂存区内。JSTL并没有提供设定cookie的动作，因为这个动作通常都是后端开发者必须去做的事情，而不是交给前端的开发者。假若我们在cookie 中设定一个名称为userCountry的值，那么可以使用${cookie.userCountry}来取得它。**

**header 和headerValues**

**header 储存用户浏览器和服务端用来沟通的数据，当用户要求服务端的网页时，会送出一个记载要求信息的标头文件，例如：用户浏览器的版本、用户计算机所设定的区域等其他相关数据。假若要取得用户浏览器的版本，即${header["User-Agent"]}。另外在鲜少机会下，有可能同一标头名称拥有不同的值，此时必须改为使用headerValues 来取得这些值。**

**注意:因为User-Agent 中包含“-”这个特殊字符，所以必须使用“[]”，而不能写成**

**$(header.User-Agent)。**

**initParam**

**就像其他属性一样，我们可以自行设定web 站台的环境参数(Context)，当我们想取得这些参数initParam就像其他属性一样，我们可以自行设定web 站台的环境参数(Context)，当我们想取得这些参数**

**<?xml version="1.0" encoding="ISO-8859-1"?>**

**<web-app xmlns="http://java.sun.com/xml/ns/j2ee"**

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

**xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"**

**version="2.4">：**

**<context-param>**

**<param-name>userid</param-name>**

**<param-value>mike</param-value>**

**</context-param>：**

**</web-app>**

**那么我们就可以直接使用 ${initParam.userid}来取得名称为userid，其值为mike 的参数。下面是之前的做法：String userid = (String)application.getInitParameter("userid");**

**param和paramValues**

**在取得用户参数时通常使用一下方法：**

**request.getParameter(String name)**

**request.getParameterValues(String name)**

**在 EL中则可以使用param和paramValues两者来取得数据。**

**${param.name}**

**${paramValues.name}**

**这里param 的功能和request.getParameter(String name)相同，而paramValues和**

**request.getParameterValues(String name)相同。如果用户填了一个表格，表格名称为username，则我们就可以使用${param.username}来取得用户填入的值。**

**看到这里，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。**

**pageContext**

**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。**

**EL算术运算**

**表达式语言支持的算术运算符和逻辑运算符非常多，所有在Java语言里支持的算术运算符，表达式语言都可以使用；甚至Java语言不支持的一些算术运算符和逻辑运算符，表达式语言也支持。**

**Code**

**<%@ page contentType="text/html; charset=gb2312"%>**

**<html>**

**<head>**

**<title>表达式语言 - 算术运算符</title>**

**</head>**

**<body>**

**<h2>表达式语言 - 算术运算符</h2>**

**<hr>**

**<table border="1" bgcolor="aaaadd">**

**<tr>**

**<td><b>表达式语言</b></td>**

**<td><b>计算结果</b></td>**

**</tr>**

**<!-- 直接输出常量 -->**

**<tr>**

**<td>\${1}</td>**

**<td>${1}</td>**

**</tr>**

**<!-- 计算加法 -->**

**<tr>**

**<td>\${1.2 + 2.3}</td>**

**<td>${1.2 + 2.3}</td>**

**</tr>**

**<!-- 计算加法 -->**

**<tr>**

**<td>\${1.2E4 + 1.4}</td>**

**<td>${1.2E4 + 1.4}</td>**

**</tr>**

**<!-- 计算减法 -->**

**<tr>**

**<td>\${-4 - 2}</td>**

**<td>${-4 - 2}</td>**

**</tr>**

**<!-- 计算乘法 -->**

**<tr>**

**<td>\${21 \* 2}</td>**

**<td>${21 \* 2}</td>**

**</tr>**

**<!-- 计算除法 -->**

**<tr>**

**<td>\${3/4}</td>**

**<td>${3/4}</td>**

**</tr>**

**<!-- 计算除法 -->**

**<tr>**

**<td>\${3 div 4}</td>**

**<td>${3 div 4}</td>**

**</tr>**

**<!-- 计算除法 -->**

**<tr>**

**<td>\${3/0}</td>**

**<td>${3/0}</td>**

**</tr>**

**<!-- 计算求余 -->**

**<tr>**

**<td>\${10%4}</td>**

**<td>${10%4}</td>**

**</tr>**

**<!-- 计算求余 -->**

**<tr>**

**<td>\${10 mod 4}</td>**

**<td>${10 mod 4}</td>**

**</tr>**

**<!-- 计算三目运算符 -->**

**<tr>**

**<td>\${(1==2) ? 3 : 4}</td>**

**<td>${(1==2) ? 3 : 4}</td>**

**</tr>**

**</table>**

**</body>**

**</html>**

**上面页面中示范了表达式语言所支持的加、减、乘、除、求余等算术运算符的功能，读者可能也发现了表达式语言还支持div、mod等运算符。而且表达式语言把所有数值都当成浮点数处理，所以3/0的实质是3.0/0.0，得到结果应该是Infinity(无穷大)。**

**如果需要在支持表达式语言的页面中正常输出“$”符号，则在“$”符号前加转义字符“\”，否则系统以为“$”是表达式语言的特殊标记。**

**EL关系运算符**

|  |  |  |  |
| --- | --- | --- | --- |
| **关系运算符** | **说明** | **范例** | **结果** |
| **== 或 eq** | **等于** | **${5==5}或${5eq5}** | **true** |
| **!= 或 ne** | **不等于** | **${5!=5}或${5ne5}** | **false** |
| **< 或 lt** | **小于** | **${3<5}或${3lt5}** | **true** |
| **> 或 gt** | **大于** | **${3>5}或{3gt5}** | **false** |
| **<= 或 le** | **小于等于** | **${3<=5}或${3le5}** | **true** |
| **>= 或 ge** | **大于等于** | **${3>=5}或${3ge5}** | **false** |

**表达式语言不仅可在数字与数字之间比较，还可在字符与字符之间比较，字符串的比较是根据其对应UNICODE值来比较大小的。**

**注意：在使用EL 关系运算符时，不能够写成：**

**${param.password1} = = ${param.password2}**

**或者**

**${ ${param.password1 } = = ${ param.password2 } }**

**而应写成**

**${ param.password1 = = param.password2 }**

**EL逻辑运算符**

|  |  |  |
| --- | --- | --- |
| **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** |

**Empty 运算符**

**Empty 运算符主要用来判断值是否为空（NULL,空字符串，空集合）。**

**条件运算符**

**${ A ? B : C}**