6

第六章

Expression Language

本章将分以下 8 节，详细介绍 Expression Language 的语法和使用：

6-1 EL 简介

EL 全名为 Expression Language，它原本是 JSTL 1.0 为方便存取数据所自定义的语言。当时 EL 只能在 JSTL 标签中使用，如下：

```
<c:out value="${ 3 + 7}">
```

程序执行结果为 10。但是你却不能直接在 JSP 网页中使用：

```
<p>Hi ! ${ username }</p>
```

到了 JSP 2.0 之后，EL 已经正式纳入成为标准规范之一。因此，只要是支持 Servlet 2.4 / JSP 2.0 的 Container，就都可以在 JSP 网页中直接使用 EL 了。

除了 JSP 2.0 建议使用 EL 之外，JavaServer Faces(JSR-127) 也考虑将 EL 纳入规范，由此可知，EL 如今已经是一项成熟、标准的技术。

注意
假若您所用的 Container 只支持 Servlet 2.3/JSP 1.2，如：Tomcat 4.1.29，您就不能在 JSP 网页中直接使用 EL，必须安装支持 Servlet 2.4 / JSP 2.0 的 Container。

6-2 EL 语法

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。

6-2-1 .与 [ ] 运算符

EL 提供 . 和 [ ] 两种运算符来存取数据。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。

接下来，我们更详细地来讨论一些情况，首先假设有一个 EL：

```
${expr-a[expr-b]}
```

(1) 当 expr-a 的值为 null 时，它会回传 null。

(2) 当 expr-b 的值为 null 时，它会回传 null。

(3) 当 expr-a 的值为一 Map 类型时：

- 假若 !value-a.containsKey(value-b)为真，则回传 null。
- 否则回传 value-a.get(value-b)。

(4) 当 expr-a 的值为 List 或 array 类型时：

- 将 value-b 的值强制转型为 int，假若不能转型为 int 时，会产生 error。
- 然后，假若 value-a.get(value-b) 或 Array.get(value-a, value-b) 产生 ArrayIndexOutOfBoundsException 或 IndexOutOfBoundsException 时，则回传 null。
- 假若 value-a.get(value-b)或 Array.get(value-a, value-b)产生其他的异常时，则会产生 error。
- 最后都没有任何异常产生时，回传 value-a.get(value-b)或 Array.get(value-a, value-b)。

(5) 当 expr-a 的值为 JavaBean 对象时：

- 将 value-b 的值强制转型为 String。
- 假若 getter 产生异常时，则会产生 error。若没有异常产生时，则回传 getter 的结果。

6-2-2 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（见表 6-1）：

表 6-1

属性范围	在 EL 中的名称
Page	PageScope
Request	RequestScope
Session	SessionScope
Application	ApplicationScope

自动搜索顺序

我们也可以指定要取出哪一个范围的变量（见表 6-2）：

表 6-2

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。有关 EL 隐含对象在 6-3 节中有更详细的介绍。

6-2-3 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。之前没接触过 JSP 的读者可能会认为上面的例子是理所当然的，但是在 JSP 1.2 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;
```

接下来再详细说明 EL 类型转换的规则：

(1) 将 A 转为 String 类型

- 假若 A 为 String 时：回传 A
- 否则，当 A 为 null 时：回传 ""
- 否则，当 A.toString( )产生异常时：错误!
- 否则，回传 A.toString( )

(2) 将 A 转为 Number 类型的 N

- 假若 A 为 null 或 "" 时：回传 0
- 假若 A 为 Character 时：将 A 转为 new Short((short)a.charValue( ))
- 假若 A 为 Boolean 时：错误!
- 假若 A 为 Number 类型和 N 一样时：回传 A
- 假若 A 为 Number 时：
    - 假若 N 是 BigInteger 时：
        - 假若 A 为 BigDecimal 时：回传 A.toBigInteger( )
        - 否则，回传 BigInteger.valueOf(A.longValue( ))
    - 假若 N 是 BigDecimal 时：
        - 假若 A 为 BigInteger 时：回传 A.toBigDecimal( )
        - 否则，回传 BigDecimal.valueOf(A.doubleValue( ))
    - 假若 N 为 Byte 时：回传 new Byte(A.byteValue( ))
    - 假若 N 为 Short 时：回传 new Short(A.shortValue( ))
    - 假若 N 为 Integer 时：回传 new Integer(A.intValue( ))
    - 假若 N 为 Long 时：回传 new Long(A.longValue( ))
    - 假若 N 为 Float 时：回传 new Float(A.floatValue( ))
    - 假若 N 为 Double 时：回传 new Double(A.doubleValue( ))
    - 否则，错误!
- 假若 A 为 String 时：
    - 假若 N 是 BigDecimal 时：
        - 假若 new BigDecimal(A)产生异常时：错误!
        - 否则，回传 new BigDecimal(A)
    - 假若 N 是 BigInteger 时：
        - 假若 new BigInteger(A)产生异常时：错误!
        - 否则，回传 new BigInteger(A)
    - 假若 N.valueOf(A)产生异常时：错误!
    - 否则，回传 N.valueOf(A)
- 否则，错误!

(3) 将 A 转为 Character 类型

- 假若 A 为 null 或 "" 时：回传 (char)0

- 假若 A 为 Character 时：回传 A
- 假若 A 为 Boolean 时：错误!
- 假若 A 为 Number 时：转换为 Short 后，然后回传 Character
- 假若 A 为 String 时：回传 A.charAt(0)
- 否则，错误!

(4) 将 A 转为 Boolean 类型

- 假若 A 为 null 或 "" 时：回传 false
- 否则，假若 A 为 Boolean 时：回传 A
- 否则,假若 A 为 String,且 Boolean.valueOf(A)没有产生异常时:回传 Boolean.valueOf(A)
- 否则，错误!

6-2-4 EL 保留字

EL 的保留字如表 6-3：

表 6-3

And	eq	gt	true
Or	ne	le	false
No	lt	ge	null
instanceof	empty	div	mod

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。

6-3 EL 隐含对象

笔者在“第五章：隐含对象（Implicit Object）”中，曾经介绍过 9 个 JSP 隐含对象，而 EL 本身也有自己的隐含对象。EL 隐含对象总共有 11 个（见表 6-4）：

表 6-4

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

续表

隐含对象	类　型	说　明
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

这 11 个隐含对象(Implicit Object)，笔者将它分成三类：

1. 与范围有关的隐含对象

- applicationScope
- sessionScope
- requestScope
- pageScope

2. 与输入有关的隐含对象

- param
- paramValues

3. 其他隐含对象

- cookie
- header
- headerValues
- initParam
- pageContext

接下来笔者会依照上面的分类顺序，为读者介绍这些隐含对象。

6-3-1 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。接下来分别对这四个隐含对象做简短的说明：

- pageScope

范围和 JSP 的 Page 相同，也就是单单一页 JSP Page 的范围(Scope)。

- requestScope

范围和 JSP 的 Request 相同，requestScope 的范围是指从一个 JSP 网页请求到另一个 JSP 网页请求之间，随后此属性就会失效。

- sessionScope

范围和 JSP Scope 中的 session 相同，它的属性范围就是用户持续在服务器连接的时间。

- applicationScope

范围和 JSP Scope 中的 application 相同，它的属性范围是从服务器一开始执行服务，到服务器关闭为止。

6-3-2 与输入有关的隐含对象

与输入有关的隐含对象有两个：param 和 paramValues，它们是 EL 中比较特别的隐含对象。一般而言，我们在取得用户的请求参数时，可以利用下列方法：

```
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。

为了让读者更加了解 param 和 paramValues 隐含对象的使用，再来看下面这个范例。此范例共有两个文件，分别为给用户输入值用的 *Param.html* 和显示出用户所传之值的 *Param.jsp*。

■ *Param.html*

```
<html>
<head>
  <title>CH6 - Param.html</title>
</head>
<body>

<h2>EL 隐含对象 param、paramValues</h2>

<form method = "post" action = "Param.jsp">
```

```
<p>姓名: <input type="text" name="username" size="15" /></p>
<p>密码: <input type="password" name="password" size="15" /></p>
<p>性别: <input type="radio" name="sex" value="Male"  checked/> 男
        <input type="radio" name="sex" value="Female"  /> 女</p>

<p>年龄:
    <select name="old">
    <option value="10">10 - 20</option>
    <option value="20" selected>20 - 30</option>
    <option value="30">30 - 40</option>
    <option value="40">40 - 50</option>
    </select>
</p>

<p>兴趣:
    <input type="checkbox" name="habit" value="Reading"/>看书
    <input type="checkbox" name="habit" value="Game"/>玩游戏
    <input type="checkbox" name="habit" value="Travel"/>旅游
    <input type="checkbox" name="habit" value="Music"/>听音乐
    <input type="checkbox" name="habit" value="Tv"/>看电视
</p>
<p>
    <input type="submit" value="传送"/>
    <input type="reset" value="清除"/>
</p>

</form>
</body>
</html>
```

Param.html 的执行结果如图 6-1 所示。当我们把窗体填好后按下传送钮，它将会把信息传送到 *Param.jsp* 做处理。

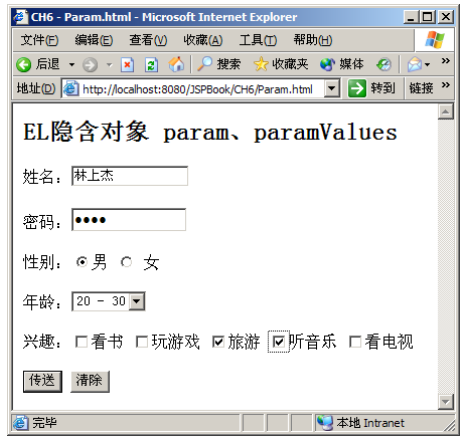

图 6-1 Param.html 的执行结果，并填入信息

接下来，*Param.jsp* 接收由 *Param.html* 传来的信息，并且将它显示出来：

■ *Param.jsp*

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<html>
<head>
  <title>CH6 - Param.jsp</title>
</head>
<body>

  <h2>EL 隐含对象 param、paramValues</h2>

  <fmt:requestEncoding value="GB2312" />

  姓名:  ${param.username}</br>
  密码:  ${param.password}</br>
  性别:  ${param.sex}</br>
  年龄:  ${param.old}</br>
  兴趣:  ${paramValues.habit[0]}
         ${paramValues.habit[1]}
</body>
</html>
```

由 *Param.html* 窗体传过来的值，我们必须指定编码方式，才能够确保 *Param.jsp* 能够顺利接收中文，传统的做法为：

```
<%
   request.setCharacterEncoding("GB2312");
%>
```

假若是使用 JSTL 写法时，必须使用 I18N 格式处理的标签库，如下：

```
<fmt:requestEncoding value="GB2312" />
```

Param.jsp 主要使用 EL 的隐含对象 param 来接收数据。但是必须注意：假若要取得多重选择的复选框的值时，必须使用 paramValues，例如：使用 paramValues 来取得“兴趣”的值，不过这里笔者最多只显示两笔“兴趣”的值：

```
${param.username}
……
${paramValues.habit[0]}
${paramValues.habit[1]}
```

有关 JSTL 的使用，第七章有更加详细的说明。图 6-2 是 *Param.jsp* 的执行结果：

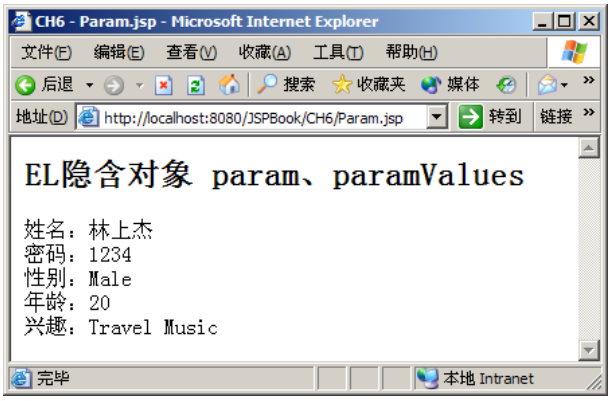

图 6-2 Param.jsp 的执行结果

6-3-3 其他隐含对象

介绍完上面六个隐含对象后，接下来将介绍最后五个隐含对象。

- 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时，可以使用 initParam 隐含对象去取得它，例如：当我们在 *web.xml* 中设定如下：

```
<?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");
```

- 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。表 6-5 列出了几个比较常用的部分。

表 6-5

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
${pageContext.request.contextPath}	服务的 web application 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

我们来看下面这个范例：*pageContext.jsp*，相信对读者来说能更加了解 pageContext 的用法。

■ *pageContext.jsp*

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
  <title>CH6 - pageContext.jsp</title>
</head>
<body>

<h2>EL 隐含对象 pageContext</h2>

\${pageContext.request.queryString}:${pageContext.request.queryString}</br>
\${pageContext.request.requestURL}:${pageContext.request.requestURL}</br>
\${pageContext.request.contextPath}:${pageContext.request.contextPath}</br>
\${pageContext.request.method}:${pageContext.request.method}</br>
```

```
\${pageContext.request.protocol}:${pageContext.request.protocol}</br>
\${pageContext.request.remoteUser}:${pageContext.request.remoteUser}</br>
\${pageContext.request.remoteAddr }:${pageContext.request.remoteAddr}</br>
\${pageContext.session.new}:${pageContext.session.new}</br>
\${pageContext.session.id}:${pageContext.session.id}</br>

</body>
</html>
```

pageContext.jsp 的执行结果如图 6-3，执行时必须在 *pageContext.jsp* 之后加上?test=1234，即 PageContext.jsp?test=1234，这样${pageContext.request.queryString}才会显示 test=1234。

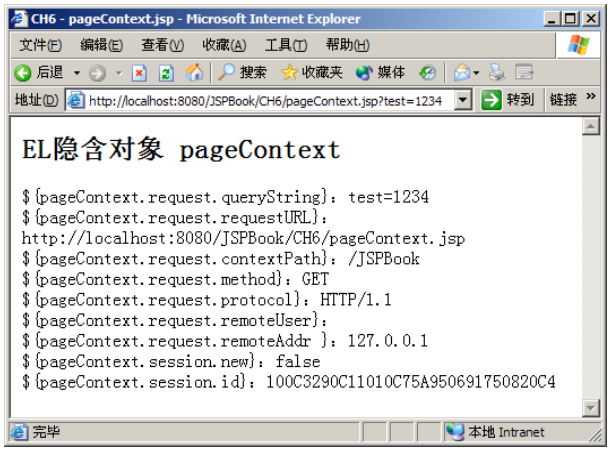

图 6-3 pageContext.jsp 的执行结果

注意
因为 ${} 在 JSP 2.0 中是特殊字符，JSP 容器会自动将它当做 EL 来执行，因此，假若要显示 ${} 时，必须在 $ 前加上 \ ，如：\${ XXXXX }

6-4 EL 算术运算符

EL 算术运算符主要有以下五个（见表 6-6）：

表 6-6

算术运算符	说 明	范 例	结 果
+	加	${ 17 + 5 }	22
—	减	${ 17 — 5 }	12
*	乘	${ 17 * 5 }	85
/ 或 div	除	${ 17 / 5 } 或 ${ 17 div 5 }	3
% 或 mod	余数	${ 17 % 5 } 或 ${ 17 mod 5 }	2

接下来，我们依照下列几种情况，详细说明 EL 算术运算符的规则：

(1) A {+ , — , *} B

- 假若 A 和 B 为 null：回传 (Long)0
- 假若 A 或 B 为 BigDecimal 时，将另一个也转为 BigDecimal，则：
  - 假若运算符为 + 时：回传 A.add(B)
  - 假若运算符为 — 时：回传 A.subtract(B)
  - 假若运算符为 * 时：回传 A.multiply(B)
- 假若 A 或 B 为 Float、Double 或包含 e / E 的字符串时：
  - 假若A或B为BigInteger时，将另一个转为BigDecimal，然后依照运算符执行运算
  - 否则，将两者皆转为 Double，然后依照运算符执行运算
- 假若 A 或 B 为 BigInteger 时，将另一个也转为 BigInteger，则：
  - 假若运算符为 + 时：回传 A.add(B)
  - 假若运算符为 — 时：回传 A.subtract(B)
  - 假若运算符为 * 时：回传 A.multiply(B)
- 否则，将 A 和 B 皆转为 Long，然后依照运算符执行运算
- 假若运算结果产生异常时，则错误!

(2) A {/ , div} B

- 假若 A 和 B 为 null：回传 (Long)0
- 假若 A 或 B 为 BigDecimal 或 BigInteger 时，皆转为 BigDecimal，然后回传 A.divide(B, BigDecimal.ROUND_HALF_UP)
- 否则，将 A 和 B 皆转为 Double，然后依照运算符执行运算
- 假若运算结果产生异常时，则错误!

(3) A {% , mod} B

- 假若 A 和 B 为 null：回传 (Long)0
- 假若 A 或 B 为 BigDecimal、Float、Double 或包含 e / E 的字符串时，皆转为 Double，然后依照运算符执行运算
- 假若 A 或 B 为 BigInteger 时，将另一个转为 BigInteger，则回传 A.remainder(B)
- 否则，将 A 和 B 皆转为 Long，然后依照运算符执行运算
- 假若运算结果产生异常时，则错误!

(4) -A

- 假若 A 为 null：回传 (Long)0
- 假若 A 为 BigDecimal 或 BigInteger 时，回传 A.negate( )
- 假若 A 为 String 时：
  - 假若 A 包含 e / E 时，将转为 Double，然后依照运算符执行运算
  - 否则，转为 Long，然后依照运算符执行运算
  - 假若运算结果产生异常时，则错误!

- 假若 A 为 Byte、Short、Integer、Long、Float 或 Double
  - 直接依原本类型执行运算
  - 假若运算结果产生异常时，则 错误!
- 否则，错误!

Tomcat 上的 *jsp-examples* 中，有一个 EL 算术运算符的范例 *basic-arithmetic.jsp*。它的程序很简单，所以不在这里多做说明，它的执行结果如图 6-4 所示。

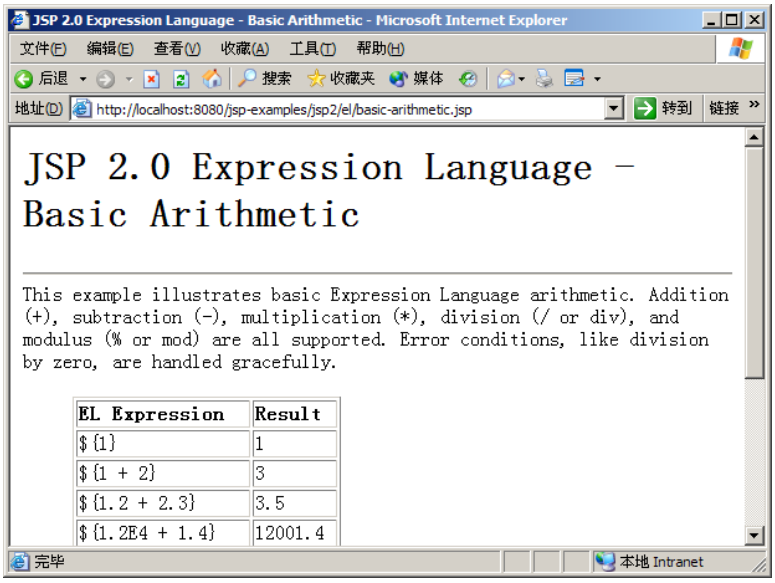

图 6-4 basic-arithmetic.jsp 的执行结果

6-5 EL 关系运算符

EL 关系运算符有以下六个运算符（见表 6-7）：

表 6-7

关系运算符	说　明	范　例	结　果
== 或 eq	等于	${5 == 5} 或 ${5 eq 5}	true
!= 或 ne	不等于	${5 != 5} 或 ${5 ne 5}	false
< 或 lt	小于	${3 < 5}或 ${3 lt 5}	true
> 或 gt	大于	${3 > 5}或 ${3 gt 5}	false
<= 或 le	小于等于	${3 <= 5}或 ${3 le 5}	true
>= 或 ge	大于等于	${3 >= 5}或 ${3 ge 5}	false

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 }`

接下来，我们依照下列几种情况，详细说明 EL 关系运算符的规则：

(1) A {<, >, <=, >=, lt, gt, le, ge} B

- 假若 A= = B，运算符为<=, le, >=, ge 时，回传 true，否则回传 false
- 假若 A 为 null 或 B 为 null 时，回传 false
- 假若 A 或 B 为 BigDecimal 时，将另一个转为 BigDecimal，然后回传 A.compareTo(B)的值
- 假若 A 或 B 为 Float、Double 时，皆转为 Double 类型，然后依其运算符运算
- 假若 A 或 B 为 BigInteger 时，将另一个转为 BigInteger，然后回传 A.compareTo(B)的值
- 假若 A 或 B 为 Byte、Short、Character、Integer 或 Long 时，皆转为 Long 类型，然后依其运算符运算
- 假若 A 或 B 为 String 时，将另一个也转为 String，然后做词汇上的比较
- 假若 A 为 Comparable 时，则：
  - 假若 A.compareTo(B)产生异常时，则错误!
- 否则，采用 A.compareTo(B) 的比较结果
- 假若 B 为 Comparable 时，则：
  - 假若 B.compareTo(A)产生异常时，则错误!
- 否则，采用 A.compareTo(B) 的比较结果
- 否则，错误!

(2) A {= =, !=, eq, ne} B

- 假若 A= = B，依其运算符运算
- 假若 A 为 null 或 B 为 null 时：= = /eq 则回传 false，!= / ne 则回传 true
- 假若 A 或 B 为 BigDecimal 时，将另一个转为 BigDecimal，则：
  - 假若运算符为 = = / eq，则 回传 A.equals(B)
  - 假若运算符为 != / ne，则 回传 !A.equals(B)
- 假若 A 或 B 为 Float、Double 时，皆转为 Double 类型，然后依其运算符运算
- 假若 A 或 B 为 BigInteger 时，将另一个转为 BigInteger，则：
  - 假若运算符为 = = / eq，则 回传 A.equals(B)
  - 假若运算符为 != / ne，则 回传 !A.equals(B)
- 假若 A 或 B 为 Byte、Short、Character、Integer 或 Long 时，皆转为 Long 类型，然后依其运算符运算

- 假若 A 或 B 为 Boolean 时，将另一个也转为 Boolean，然后依其运算符运算
- 假若 A 或 B 为 String 时，将另一个也转为 String，然后做词汇上的比较
- 否则，假若 A.equals(B)产生异常时，则 错误!
- 否则，然后依其运算符运算，回传 A.equals(B)

Tomcat 上的 jsp-examples 中，有一个 EL 关系运算符的范例 *basic-comparisons.jsp*。它的程序很简单，所以不在这里多做说明，大家直接看它的执行结果（如图 6-5 所示）：

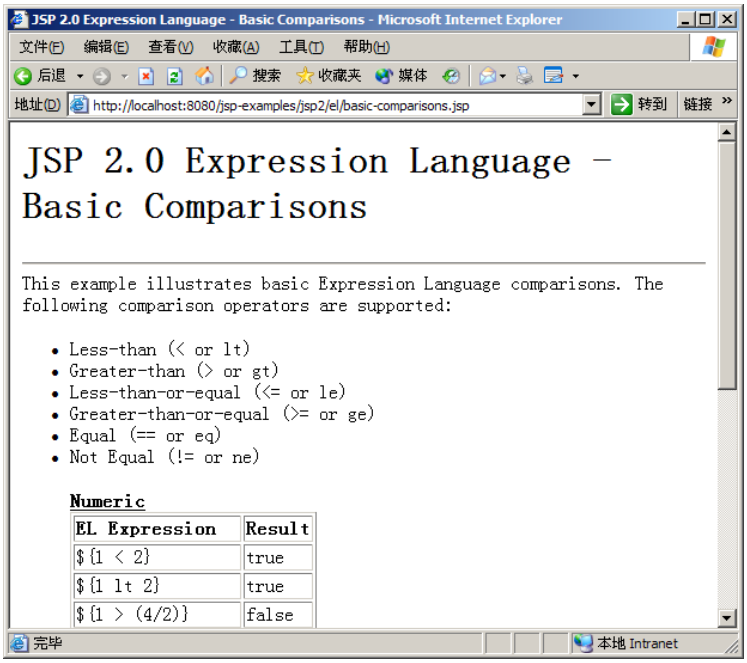

图 6-5 basic-comparisons.jsp 的执行结果

6-6 EL 逻辑运算符

EL 逻辑运算符只有三个（见表 6-8）：

表 6-8

逻辑运算符	说 明	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

下面举几个例子：

```
${ param.month = = 7 and param.day = = 14 }
${ param.month = = 7 || param.day = = 14 }
${ not param.choice }
```

EL 逻辑运算符的规则很简单：

(1) A {&&, and, || 或 or } B
- 将 A 和 B 转为 Boolean，然后依其运算符运算

(2) {!, not}A
- 将 A 转为 Boolean，然后依其运算符运算

6-7 EL 其他运算符

EL 除了上述三大类的运算符之外，还有下列几个重要的运算符：

(1) Empty 运算符
(2) 条件运算符
(3) ( ) 括号运算符

6-7-1 Empty 运算符

Empty 运算符主要用来判断值是否为 null 或空的，例如：

```
${ empty param.name }
```

接下来说明 Empty 运算符的规则：

(1) {empty} A
- 假若 A 为 null 时，回传 true
- 否则，假若 A 为空 String 时，回传 true
- 否则，假若 A 为空 Array 时，回传 true
- 否则，假若 A 为空 Map 时，回传 true
- 否则，假若 A 为空 Collection 时，回传 true
- 否则，回传 false

6-7-2 条件运算符

所谓条件运算符如下：

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

意思是说，当 A 为 true 时，执行 B；而 A 为 false 时，则执行 C。

6-7-3 括号运算符

括号运算符主要用来改变执行优先权，例如：${ A * (B+C) }

至于运算符的优先权，如下所示(由高至低，由左至右)：

- []、.
- ( )
- - (负)、not、!、empty
- *、/、div、%、mod
- +、- (减)
- <、>、<=、>=、lt、gt、le、ge
- = =、!=、eq、ne
- &&、and
- ||、or
- ${ A ? B : C}

最后笔者写一个 *ELOperator.jsp* 范例，将所有运算符实际操作一遍。

■ *ELOperator.jsp*

```
<%@ page contentType="text/html;charset=GB2312" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
<title>CH6 – ELOperator.jsp</title>
</head>
<body>

<h2>EL 的运算符</h2>

<c:set value="mike" var="username" scope="request" />
<table border="1" width="50%" align="left">
<TR>
  <TR>
   <TH>运算式</TH>
   <TH>结果</TH>
  </TR>
  <TR><TD>14 + 3</TD><TD>${14 + 3}</TD></TR>
  <TR><TD>14 - 3</TD><TD>${14 - 3}</TD></TR>
  <TR><TD>14 * 3</TD><TD>${14 * 3}</TD></TR>
  <TR><TD>14 / 3</TD><TD>${14 / 3}</TD></TR>
  <TR><TD>14 % 3</TD><TD>${14 % 3}</TD></TR>
  <TR><TD>14 == 3</TD><TD>${14 == 3}</TD></TR>
  <TR><TD>14 != 3</TD><TD>${14 != 3}</TD></TR>
  <TR><TD>14 < 3</TD><TD>${14 < 3}</TD></TR>
  <TR><TD>14 > 3</TD><TD>${14 > 3}</TD></TR>
  <TR><TD>14 <= 3</TD><TD>${14 <= 3}</TD></TR>
```

```
  <TR><TD>14 >= 3</TD><TD>${14 >= 3}</TD></TR>
  <TR><TD>true && false</TD><TD>${true && false}</TD></TR>
  <TR><TD>true || false</TD><TD>${true || false}</TD></TR>
  <TR><TD>! false</TD><TD>${! false}</TD></TR>
  <TR><TD>empty username</TD><TD>${empty username}</TD></TR>
  <TR><TD>empty password</TD><TD>${empty password}</TD></TR>
</table>
</body>
</html>
```

EL 的数学运算符、相等运算符、关系运算符和逻辑运算符就跟其他程序语言一样，并没有特别的地方。但是它的 empty 运算符就比较特别，为了测试它，笔者写了这样一行程序代码：

```
<c:set value="mike" var="username" scope="request" />
```

这样 Request 属性范围里就存在一个名称为 username、值为 mike 的属性。执行此程序时，读者将会发现${empty username}为 false； ${empty password}为 true，其代表的意义就是：它可以在四种属性范围中找到 username 这个属性，但是找不到 password 这个属性。*ELOperator.jsp* 的执行结果如图 6-6：

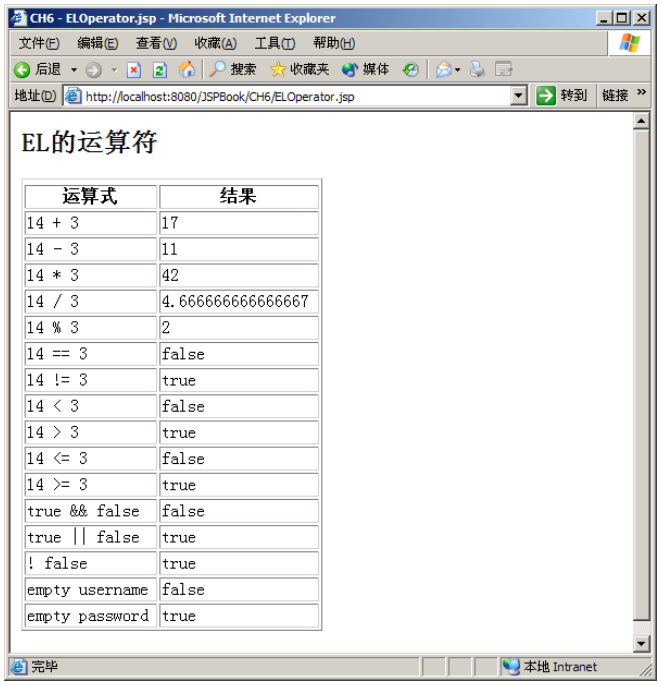

图 6-6 ELOperator.jsp 的执行结果

6-8 EL Functions

前面几节主要介绍 EL 语法的使用和规则，本节笔者将介绍如何自定义 EL 的函数(functions)。

EL 函数的语法如下：

```
ns:function( arg1, arg2, arg3 …. argN)
```

其中 ns 为前置名称(prefix)，它必须和 taglib 指令的前置名称一样。如下范例：

```
<% @ taglib prefix="my"
  uri="http://jakarta.apache.org/tomcat/jsp2-example-taglib" %>
…….
${my:function(param.name)}
```

前置名称都为 my，至于 function 为 EL 函数的名称，而 arg1、arg2 等等，都是 function 的传入值。在 Tomcat 5.0.16 中有一个简单的 EL 函数范例，名称为 *functions.jsp*，笔者接下来将依此范例来说明如何自定义 EL 函数。

6-8-1 Tomcat EL 函数范例

Tomcat 提供的 EL 函数范例中，自定义两个 EL 函数：reverse 和 countVowels，其中：

reverse 函数：将传入的字符串以反向顺序输出。
countVowels 函数：计算传入的字符串中，和 aeiouAEIOU 吻合的字符个数。

图 6-7 是 *functions.jsp* 程序的执行结果：

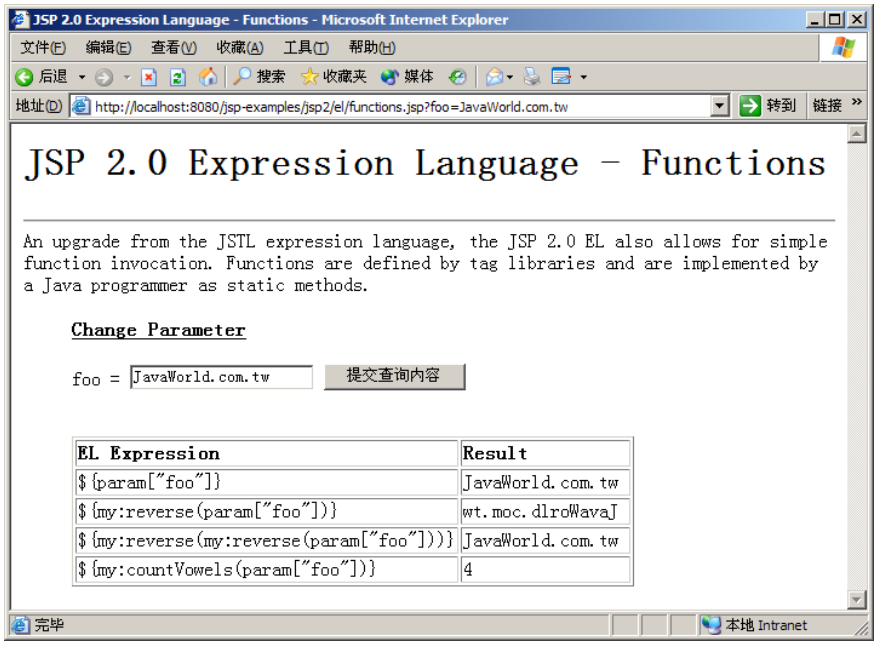

图 6-7 functions.jsp 的执行结果

输入 JavaWorld.com.tw 字符串至 reverse 函数后，回传 wt.moc.dlroWavaJ 的结果；若

传入 countVowels 函数后，因为有两个 a 和 o，总共四个字符吻合，所以回传 4。

Tomcat 的 EL 函数范例，主要分为四个部分（见表 6-9）：

表 6-9

web.xml	设定 taglib 的 TLD 文件位置
functions.jsp	使用 EL 函数的范例程序
jsp2-example-taglib.tld	EL 函数、标签库的设定文件
jsp2.examples.el.Functions.java	EL 函数主要程序逻辑处理部分

这四个部分环环相扣，都互有关系，笔者依 *functions.jsp* 为中心，然后再慢慢说明其他部分。首先我们直接来看 *functions.jsp* 程序：

6-8-2 functions.jsp

■ *functions.jsp*

```
<%@ taglib prefix="my"
uri="http://jakarta.apache.org/tomcat/jsp2-example-taglib"%>
<html>
  <head>
    <title>JSP 2.0 Expression Language - Functions</title>
  </head>
  <body>
    <h1>JSP 2.0 Expression Language - Functions</h1>
    …. 略
    <blockquote>
      <u><b>Change Parameter</b></u>
      <form action="functions.jsp" method="GET">
       foo = <input type="text" name="foo" value="${param['foo']}">
          <input type="submit">
      </form>
      <br>
      <code>
        <table border="1">
          <thead>
        <td><b>EL Expression</b></td>
        <td><b>Result</b></td>
      </thead>
      <tr>
         <td>\${param["foo"]}</td>
         <td>${param["foo"]} </td>
      </tr>
      <tr>
         <td>\${my:reverse(param["foo"])}</td>
         <td>${my:reverse(param["foo"])} </td>
      </tr>
      <tr>
         <td>\${my:reverse(my:reverse(param["foo"]))}</td>
```

```
        <td>${my:reverse(my:reverse(param["foo"]))} </td>
      </tr>
      <tr>
        <td>\${my:countVowels(param["foo"])}</td>
        <td>${my:countVowels(param["foo"])} </td>
       </tr>
     </table>
     </code>
   </blockquote>
  </body>
</html>
```

*functions.jsp*程序中，一开始定义taglib，它的前置名称为my；uri为http://jakarta.apache.org/tomcat/jsp2-example-taglib，如下所示：

```
<%@ taglib prefix="my"
uri="http://jakarta.apache.org/tomcat/jsp2-example-taglib"%>
```

当Container执行这段程序时，它会根据uri的值，到*web.xml*中找相对应的TLD (Tag Library Descriptor)文件。至于 *web.xml* 如何设定两者之间的对应关系，我们在 6-8-3 小节再说明。

functions.jsp 中包含一个窗体(form)，当用户在文本输入框(text input)中输入字符串，按下按钮时，底下会显示字符串经过 EL 函数处理后的结果。*functions.jsp* 程序最重要的部分是调用 EL 函数：

```
${my:reverse(param["foo"])}
```

上述的意思是接收 foo 参数，然后传入 reverse 函数。调用 EL 函数的方式很简单，只要前置名称：其中 EL 函数名称是被定义在 TLD 文件中，这会在 6-8-4 小节详细说明。至于 reverse 函数的逻辑运算，则是被定义在 *jsp2.examples.el.Functions.java* 程序中，这部分会在 6-8-5 小节中说明。

注意
TLD 文件主要为标签的设定文件，其中包含标签的名称、参数等等。在 JSP 2.0 之后，相关 EL 函数的设定，也可以在 TLD 文件中定义。

6-8-3 web.xml

web.xml 是每个 web 站台最主要的设定文件，在这个设定文件中，可以设定许多东西，如：Servlet、Resource、Filter 等等。不过现在关心的是如何在 *web.xml* 中设定 taglib 的 uri 是对应到哪个 TLD 文件。笔者从范例的 *web.xml* 中节录出设定的片段程序如下：

■ *web.xml*

```
<jsp-config>
  <taglib>
    <taglib-uri>
        http://jakarta.apache.org/tomcat/jsp2-example-taglib
    </taglib-uri>
    <taglib-location>
```

```
      /WEB-INF/jsp2/jsp2-example-taglib.tld
    </taglib-location>
  </taglib>
</jsp-config>
```

在 *web.xml* 中，<taglib>用来设定标签的 TLD 文件位置。<taglib-uri>用来指定 taglib 的 uri 位置，用户可以自行给定一个 uri，例如：

```
<taglib-uri>http://www.javaworld.com.tw/jute</taglib-uri>
<taglib-uri>tw.com.javaworld</taglib-uri>
```

<taglib-location>用来指定 TLD 文件的位置。依照范例，它是指定在 *WEB-INF/jsp2/*目录下的 *jsp2-example-taglib.tld*。

因此，笔者所节录下来的 *web.xml*，它所代表的意思是：taglib 的 uri 为 http://jakarta.apache.org/tomcat/jsp2-example-taglib，它的 TLD 文件是在 *WEB-INF/jsp2/* 目录下的 *jsp2-example-taglib.tld*。

6-8-4 jsp2-example-taglib.tld

在 *jsp2-example-taglib.tld* 中定义许多标签，其中笔者节录一段定义 EL 函数：

■ *jsp2-example-taglib.tld*

```
<function>
    <description>Reverses the characters in the given String</description>
    <name>reverse</name>
    <function-class>jsp2.examples.el.Functions</function-class>
    <function-signature>
      java.lang.String reverse( java.lang.String )
    </function-signature>
</function>
<function>
    <description>Counts the number of vowels (a,e,i,o,u) in the given
      String</description>
    <name>countVowels</name>
    <function-class>jsp2.examples.el.Functions</function-class>
    <function-signature>
      java.lang.String numVowels( java.lang.String )
    </function-signature>
</function>
```

上述定义两个 EL 函数，用<name>来设定 EL 函数名称，它们分别为 reverse 和 countVowels；用<function-class>设定 EL 函数的 Java 类，本范例的 EL 函数都是定义在 *jsp2.examples.el.Functions*；最后用<function-signature>来设定 EL 函数的传入值和回传值，例如：

```
<function-signature>java.lang.String
   reverse( java.lang.String )</function-signature>
```

表示 reverse 函数有一 String 类型的传入值，然后回传 String 类型的值。最后我们再来看

reverse 和 countVowels 的程序。

6-8-5 Functions.java

Functions.java 主要定义三个**公开静态**的方法，分别为：reverse、numVowels 和 caps（见表 6-10）。下面是 *Functions.java* 完整的程序代码：

■ *Functions.java*

```
package jsp2.examples.el;

import java.util.*;

/**
 * Defines the functions for the jsp2 example tag library.
 * <p>Each function is defined as a static method.</p>
 */
public class Functions {
    public static String reverse( String text ) {
        return new StringBuffer( text ).reverse().toString();
    }

    public static int numVowels( String text ) {
        String vowels = "aeiouAEIOU";
        int result = 0;
        for( int i = 0; i < text.length(); i++ ) {

            if( vowels.indexOf( text.charAt( i ) ) != -1 ) {
              result++;
            }
         }
         return result;
    }

    public static String caps( String text ) {
        return text.toUpperCase();
    }
}
```

表 6-10

String reverse(String text)	将 text 字符串的顺序反向处理，然后回传反向后的字符串
int numVowels(String text)	将 text 字符串比对 aeiouAEIOU 等字符，然后回传比对中的次数
String caps(String text)	将 text 字符串都转为大写，然后回传此字符串

注意
在定义 EL 函数时，都必须为**公开静态**(public static)