第 18 章　EL表达式语言

EL（Expression Language）表达式语言是属于 JSP2.0 版本提出了一个新的内容。其实，读者在前几章中已经有了初步的接触，并且知道它可以很好的和 JSTL 或者自定义标签结合使用，也可以在脚本语言中使用。这一章将重点地向读者讲解 EL 的具体使用方法。

本章要点包括以下内容：

- EL 表达式语言简介
- EL 表达式的语法
- EL 表达式中的隐含对象
- EL 中调用自定义函数

18.1　EL表达式语言简介

起初 EL 表达式只是为了方便存取数据所定义的一种语言。它只能使用在 JSTL 标签中，而不能在 JSP 页面中运用。一直到了 JSP2.0 版本发布后，EL 才被正式纳入为 JSP 标准规范之一。这时，它才可以直接在 JSP 页面程序中使用，只要安装的 Web 服务器能够支持 Servlet2.4/JSP2.0。

18.1.1　EL表达式简单使用介绍

在 JSP2.0 之前，Web 开发者只能使用表达式<%=Name%>访问系统的值，例如：

```
<someTags:aTag attribute="<%=pageContext.getAttribute("aName")%>">
```

或者调用 JavaBean 中的属性值：

```
<%= aCustomer.getAddress().getCountry() %>
```

而 EL 表达式语言允许网页开发者使用简单的语法访问对象。比如要访问一个简单变量，可以像下面这样写：

```
<someTags:aTag attribute="${aName}">
```

其中${aName}即为访问 aName 变量的 EL 表达式。而访问 JavaBeans 属性（例如获取 aCustomer 对象中定义的 address 对象变量的 country 属性），可以使用：

```
${aCustomer.address.country}
```

EL 表达式还可以很好在 JavaScript 中得到使用，因为表达式语言正是借用了访问结构化数据的 JavaScript 语法。EL 可以很好的和 JSTL 或者自定义标签结合使用，因为 EL 创建的初衷就是在 JSTL 中使用。例如，在<c:if>标签中的不等式 EL：

```
<c:if test=”${bean1.num < 3}”>
   body content
</c:if>
```

在 EL 中除了可以是变量、数值、对象的属性调用或者嵌套调用以及不等式判断（判断结果是 Boolean 类型），还可以直接包含一个字符串：

```
<c:out value=”an expression is ${‘${‘}expr}” />
```

这样输出的字符串值为：an expression is ${expr}。

在前面章节讲解 page 指令时，提到的一个 isELignored 属性，它就是指定该 JSP 页面是否支持 EL 表达式。如果 isELignored 属性值为 true，即忽略 EL 表达，在 JSP 页面可以直接使用${之类的字符，Web 容器式不会试图解析这些表达式。如果设置为 false，当 Web 容器遇到“${”字符时会解析其中的表达式内容，并把结果输出。

注意：当 isELignored 属性设置为 false，即页面能够识别 EL 表达式语言，但是 Web 开发者又想使用“${”字符，则需要在前面加上“/”作标识。

18.1.2　EL表达式的特点和使用范围

总结起来，EL 表达式语言的使用范围以及特点包括如下：

- 不仅仅可以在 JSTL 或者自定义标签中使用 EL，JSP2.0 开始允许 EL 在 JSP 语句以及 JavaScript 语句等更广范围内使用 EL 表达式语言。
- 在 EL 表达式语言中可以获得命名空间（PageContext 对象，它是页面中所有其他内置对象的最大集成者，通过它可以访问到其他内置对象，本书的第五章有介绍）。
- 既可以访问一般变量，还可以访问 JavaBean 类中的属性以及嵌套属性和集合对象。
- 在 EL 表达式中可以执行关系运算、逻辑运算以及算术运算。
- 扩展函数可以和 Java 类的静态方法映射
- 在 EL 表达式中可以访问 JSP 的一系列隐含对象（request、session、application 以及 page 等）。

18.2　EL语法

EL 表达式的使用是非常简单的，所有的 EL 表达式的格式都是以“${”开始，并以“}”结尾。最直接和简单的方法，就是在 EL 中使用变量名获取到值，例如：

```
${ username }
```

当 EL 表达式中的变量不给定范围时，则表示容器会默认从 page 范围中找，再依次到 request.、session 以及 application 范围。如果中途中找到 username 变量，则直接返回，否则返回 NULL。下面列出了 EL 使用到的变量属性范围的名称。

- 属性范围 page：在 EL 中使用名称 pageScope，例如${pageScope.username}，则表示在 page 范围中找 username 变量，找不到直接返回 NULL。
- 属性范围 request：在 EL 中使用名称 requestScope。
- 属性范围 session：在 EL 中使用名称 sessionScope。
- 属性范围 application：在 EL 中使用名称 applicationScope。

18.2.1　在EL中可以包含的文字

JSP 表达式语言（EL）定义了可以在表达式中使用的文字包括以下：

- Boolean：ture 和 false。
- Integer：与 Java 类似，可以包含任何正数或者负数，例如 24、56、-54 等。
- Floating Point：与 Java 类似，可以包含任何正的或者负的浮点数，例如-1.8E-5、4.567。

- String：任何由引号或者双引号限定的字符串。对于单引号、双引号和反斜杠，使用反斜杠字符作为转义序列。必须注意，如果在字符串两端使用双引号，则但引号不需要转义。
- NULL：返回 null。

下面逐一举例：

${false}包含 Boolean 类型、${3*8}包含整数类型 Integer、${5.678}包含浮点型、${“string”}或者${‘string’}包含字符串、${NULL}表示返回为空值。

18.2.2　操作符“[]”和“.”

在 EL 中，可以使用操作符“[]”和“.”来取得对象的属性。例如：

${user.name}或者${user[name]}表示取出对象 user 中的 name 属性值。

另外在 EL 中可以使用[]操作符来读取 Map、List 或者对象集合中的数据，例如在 Session 域中存在一个数组 schools：获取数组中第二位数值：

```
${sessionScope.schools[1]}
```

如果 schools 是 School 对象集合类，则可是获取第二位对象中的 name 属性值：

```
${sessionScope.shools[1].name}
```

这里的 sessionScope 是在 EL 表达式中的调用 Session 内置对象的名称。

如果需要 EL 表达式来访问一个 JavaBean 中的属性值，JavaBean 的定义如下：

```
<jsp:useBean id=”user” class=”cn.User” />
```

对 JavaBean 类中属性的引用：${user.name}或者${user[“name”]}。

注意：当属性名中包含了一些特殊符号时，例如“.”或者“ - ”等非字母或者数字符号时，就只能使用[]格式来访问属性值了，例如：

${sessionScope.user[user_name]}**是正确的，而**${sessionScope.user.user_name}**是不正确的。**

另外一种情况也必须使用[]符号来引用属性值，当某个对象的属性名使用某一个变量（假设这里变量为 AttributeName **）来给定的，例如：**

${sessionScope.user[AttributeName]}，**这里的变量** AttributeName **可以在之前进行赋值。例如** AttributeName **变量赋值为** name**，则以上** EL **表达式等价于：**${sessionScope.user.name}**。**

18.2.3　EL中的基本算术操作符

在 JSP 中的表达式语言（EL）提供了如下多个运算操作符，其中大部分是 Java 中常用的操作符，如表 18.1 所示。

表 18.1　五种算术操作符

算术操作符	说明	范例	结果
＋	“加法”操作	${23+5}	28
－	“减法”操作	${34-6}	28
*	“乘法”操作	${3*8}	24
/或者div	“除法”操作	${8/2}或者${8 div 2}	4
%或者mod	“求余”操作	${17/4}或者${17 mod 4}	1

接下来对这五个操作符运算作详细讲解。

18.2.3.1　“＋”、“－”和“*”操作符：A＋B、A－B 和 A*B

- 如果 A 和 B 同为 NULL 时，直接返回 0。
- 如果 A 和 B 其中有一个为 Float、Double 或者包含了“.”、“e”或者“E”字符的字符串，则强行把 A 和 B 转换成 Double 类型，其中有三种情况：

（1）运算符为“＋”时，返回 A.add(B)；

（2）运算符为“－”，则返回 A.subtract(B)；

（3）运算符为“*”，则返回 A.multiply(B)。

- 否则，强行把 A 和 B 转为 Long 类型，然后再运行相应的运算。
- 如果运算操作出现异常，则报错。

18.2.3.2　“/”或者 div 操作符：A/B 或者 A div B

- 如果 A 和 B 同时为 NULL，则返回 0
- 否则，强行把 A 和 B 转换成 Double 类型，再执行相应的操作运算
- 如果运算操作出现异常，则报错

18.2.3.3　“%”或者“mod”操作符：A%B 或者 A mod B

- 如果 A 和 B 同时为 NULL，则返回 0
- 如果 A 或者 B 其中一个为 Float、Double 或者包含了“.”、“e”或者“E”字符的字符串，则强行把 A 和 B 转换成 Double 类型，再执行相应的操作运算
- 否则，强行把 A 和 B 转换成 Long 类型，再执行相应操作运算
- 如果运算操作出现异常，则报错

18.2.3.4　一元负号操作符：-A

- 如果 A 为 NULL，则返回 0
- 如果 A 为一个字符串，又分如下几种情况：

（1）如果这个字符串包含“.”、“e”或者“E”字符，则把 A 强行转换成 Double 类型，再执行相应操作运算；

（2）否则，强行把 A 转换成 Long 类型，再执行相应操作运算；

（3）如果运算操作出现异常，则报错。

- 如果 A 是一个 Byte、Short、Integer、Long、Float 或者 Double 类型时，同样可能出现两种情况：

（1）直接按照原来的类型执行相应的操作运算；

（2）如果出现操作异常，则报错。

- 否则，出现异常，报错

18.2.4　EL的基本算术操作实例

介绍了这五个操作符的运算法则，下面列举一个例子来体验一下，创建一个 basic_arithmetic.jsp 文件（重新创建一个名为 MyEL 的 J2EE 项目，Web 模块名为 EL），其源代码如下：

```
<%@ page isELIgnored="false" %>
<html>
  <head>
    <title>Basic Arithmetic</title>
  </head>
  <body>
    <h3>EL Basic Arithmetic: "+","-","*","/ or div","% or mod"</h3>
    <hr>
```

```
<br>
<blockquote>
  <code>
    <table border="1">
      <thead>
     <td><b>EL Expression</b></td>
     <td><b>Result</b></td>
   </thead>
   <tr>
     <td>\${1}</td>
     <td>${1}</td>
   </tr>
   <tr>
     <td>\${1 + 2}</td>
     <td>${1 + 2}</td>
   </tr>
   <tr>
     <td>\${1.2 + 2.3}</td>
     <td>${1.2 + 2.3}</td>
   </tr>
   <tr>
     <td>\${1.2E4 + 1.4}</td>
     <td>${1.2E4 + 1.4}</td>
   </tr>
   <tr>
     <td>\${-4 - 2}</td>
     <td>${-4 - 2}</td>
   </tr>
   <tr>
     <td>\${21 * 2}</td>
     <td>${21 * 2}</td>
   </tr>
   <tr>
     <td>\${3/4}</td>
     <td>${3/4}</td>
   </tr>
   <tr>
     <td>\${3 div 4}</td>
     <td>${3 div 4}</td>
   </tr>
   <tr>
     <td>\${3/0}</td>
     <td>${3/0}</td>
   </tr>
   <tr>
     <td>\${10%4}</td>
     <td>${10%4}</td>
   </tr>
   <tr>
     <td>\${10 mod 4}</td>
```

```
            <td>${10 mod 4}</td>
          </tr>
        <tr>
          <td>\${(1==2) ? 3 : 4}</td>
          <td>${(1==2) ? 3 : 4}</td>
        </tr>
       </table>
          </code>
        </blockquote>
    </body>
</html>
```

程序说明：以上实例列举了五种操作符的实际使用方法，该页面的运行结果如图 18.1 所示。

注意：如果页面没有支持 EL 表达式语言，则在 JSP 头部一定要加入<%@ page isELIgnored="false" %>**语句。**

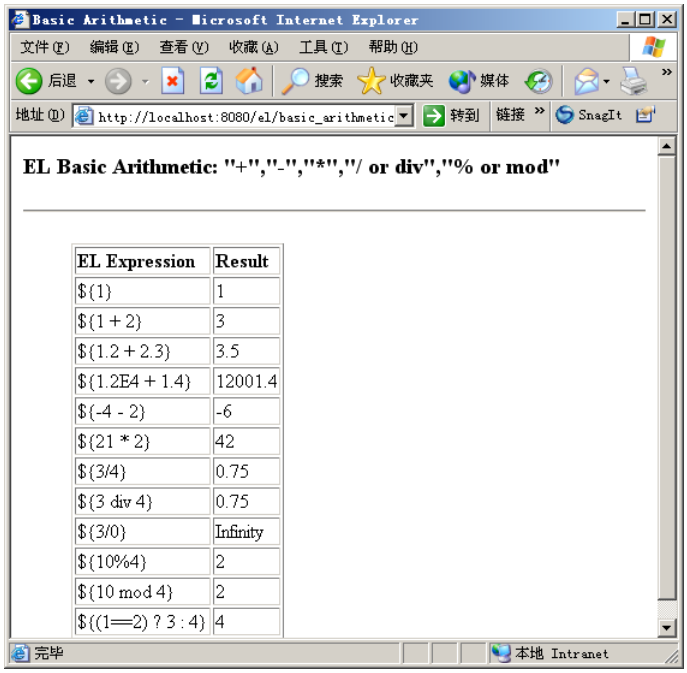

图 18.1　五种基本运算符的运算结果

18.2.5　EL中的关系操作符

EL 中使用的关系操作包括如下六种，如表 18.2 所示。接下来对这些操作符作依次讲解。

表 18.2　关系操作符

关系操作符	说明	范例	结果
= =或者eq	“等于”判断	${5= =10}或者${5 eq 10}	false
!=或者ne	“不等于”判断	${5 != 10}或者${5 ne 10}	true
<或者lt	“小于”判断	${3 < 5}或者${3 lt 5}	true
>或者gt	“大于”判断	${3 > 5}或者${3 gt 5}	false
<=或者le	“小于等于”判断	${4 <= 4}或者${4 le 4}	true
>=或者ge	“大于等于”判断	${4 >= 6}或者${4 ge 6}	false

18.2.5.1　A{< or lt、> or gt、<= or le、>= or ge}B 操作符

- 如果 A 值等于 B 时，操作符为“<=”或者“le”、“>=”或者“ge”，则返回 true。否则，操作符为“<”或者“lt”、“>”或者“gt”时，返回 false
- 如果 A 是 NULL，或者 B 是 NULL，则返回 false
- 如果 A 或者 B 是 Float 或者 Double 类型，则强行把 A 和 B 转换成 Double 类型，然后再执行相应的操作运算符
- 如果 A 或者 B 是 Byte、Short、Character、Integer 或者 Long 类型时，则强行把 A 和 B 转换成 Long，然后执行相应的操作运算符。
- 如果 A 或者 B 是一个字符串，则强行把 A 和 B 全都转换成字符串，再进行相应的比较操作。
- 如果拿 A 是去比较，存在两种情况：

（1）若 A.compareTo(B)产生异常，则报错；

（2）否则，返回 A.compareTo(B)的比较结果。

- 如果拿 B 是去比较，也存在两种情况：

（1）若 B.compareTo(A)产生异常，则报错；

（2）否则，返回 B.compareTo(A)的比较结果。

- 否则，产生异常，报错

18.2.5.2　A{== or eq、!= or ne}B 操作符

- 如果 A 等于 B，操作符为“＝＝”或者“eq”时，返回 true；否则，操作符为“！＝”或者“ne”时，返回 false
- 如果 A 为 NULL 或者 B 为 NULL 时，则对操作符“＝＝”或者“eq”，返回 false；如果操作符是“！＝”或者“ne”，则返回 true
- 如果 A 或者 B 是 Float 或者 Double 类型，则强行把 A 和 B 转换成 Double 类型，然后再执行相应的操作运算符
- 如果 A 或者 B 是 Byte、Short、Character、Integer 或者 Long 类型时，则强行把 A 和 B 转换成 Long 类型，然后再执行相应的操作运算符
- 如果 A 或者 B 是 Boolean 类型，则强行把 A 和 B 转换成 Boolean 类型之后再执行相应的操作运算府
- 如果 A 或者 B 是字符串类型，则强行把 A 和 B 转换成字符串类型，再执行两者之间的比较
- 如果在执行 A.equals(B)时产生异常，报错，否则返回比较结果。

18.2.6　EL的关系操作实例

同样，引用 Tomcat 服务器自带的实例，来检验这些操作运算符的执行效果，创建的 basic_comparisons.jsp 文件代码如下：

```
<%@ page isELIgnored="false" %>
<html>
<head><title>Expression Language - Basic Comparisons</title></head>
<body>
<h4>EL - Basic Comparisons: "==","!=","<",">","<=",">="</h4>
<hr>
<table>
  <tr>
```

```
<td>
    <ul>
      <li>Less-than (&lt; or lt)</li>
      <li>Greater-than (&gt; or gt)</li>
      <li>Less-than-or-equal (&lt;= or le)</li>
      <li>Greater-than-or-equal (&gt;= or ge)</li>
      <li>Equal (== or eq)</li>
      <li>Not Equal (!= or ne)</li>
    </ul>
  </td>
  <td>
    <blockquote>
      <u><b>Numeric</b></u>
      <code>
      <table border="1">
        <thead>
                <td><b>EL Expression</b></td>
                <td><b>Result</b></td>
            </thead>
                <tr>
                  <td>\${1 &lt; 2}</td>
                  <td>${1 < 2}</td>
                </tr>
                <tr>
                  <td>\${1 lt 2}</td>
                  <td>${1 lt 2}</td>
                </tr>
                <tr>
                  <td>\${1 &gt; (4/2)}</td>
                  <td>${1 > (4/2)}</td>
                </tr>
                <tr>
                  <td>\${1 &gt; (4/2)}</td>
                  <td>${1 > (4/2)}</td>
                </tr>
                <tr>
                  <td>\${4.0 &gt;= 3}</td>
                  <td>${4.0 >= 3}</td>
                </tr>
                <tr>
                  <td>\${4.0 ge 3}</td>
                  <td>${4.0 ge 3}</td>
                </tr>
                <tr>
                  <td>\${4 &lt;= 3}</td>
                  <td>${4 <= 3}</td>
                </tr>
                <tr>
                  <td>\${4 le 3}</td>
                  <td>${4 le 3}</td>
```

```
                        </tr>
                        <tr>
                           <td>\${100.0 == 100}</td>
                           <td>${100.0 == 100}</td>
                        </tr>
                        <tr>
                           <td>\${100.0 eq 100}</td>
                           <td>${100.0 eq 100}</td>
                        </tr>
                        <tr>
                           <td>\${(10*10) != 100}</td>
                           <td>${(10*10) != 100}</td>
                        </tr>
                        <tr>
                           <td>\${(10*10) ne 100}</td>
                           <td>${(10*10) ne 100}</td>
                        </tr>
                     </table>
            </code>
          </blockquote>
        </td>
        <td>
       <blockquote>
       <u><b>Alphabetic</b></u>
       <code>
          <table border="1">
             <thead>
                           <td><b>EL Expression</b></td>
                           <td><b>Result</b></td>
                        </thead>
                        <tr>
                           <td>\${'a' &lt; 'b'}</td>
                           <td>${'a' < 'b'}</td>
                        </tr>
                        <tr>
                           <td>\${'hip' &gt; 'hit'}</td>
                           <td>${'hip' > 'hit'}</td>
                        </tr>
                        <tr>
                           <td>\${'4' &gt; 3}</td>
                           <td>${'4' > 3}</td>
                        </tr>
                     </table>
       </code>
     </blockquote>
        </td>
      </tr>
</table>
</body>
</html>
```

该页面的运行效果如图 18.2 所示。

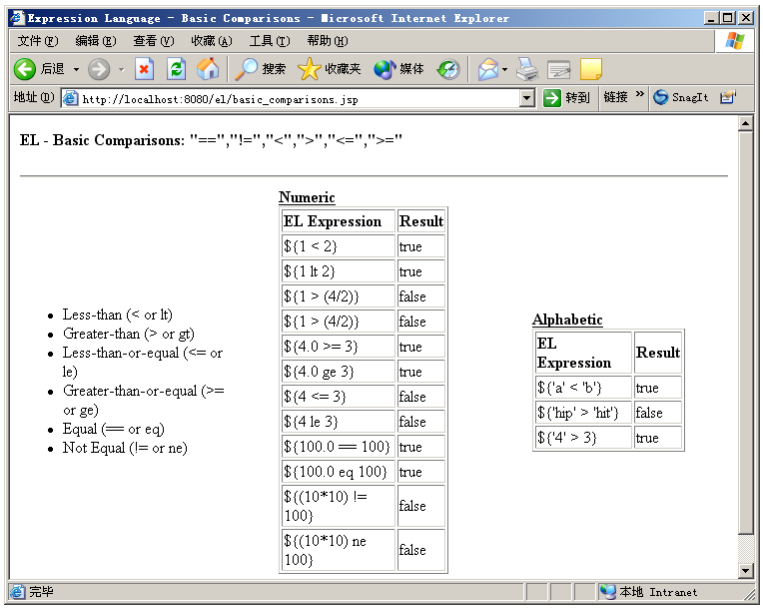

图 18.2　关系操作符页面的运行效果

18.2.7　逻辑操作符

逻辑操作符包括如下几种，如表 18.3 所示。

表 18.3　逻辑操作符

逻辑操作符	说明	范例
&& 或者 and	“与”操作符	${num >5 && num <10}
|| 或者 or	“或”操作符	${num <5 || num > 11}
! 或者 not	“非”操作符	${!name.equals(“zzb”)}

18.2.7.1　A{&& 或者 and、|| 或者 or}B 逻辑操作

执行该操作符时，首先强行把 A 和 B 转换成 Boolean 类型，然后再执行相应的操作运算符。

- ❑ 如果 A 和 B 同为 true，当操作符为“&&”时，返回 true；操作符为“||”时，返回 true。
- ❑ 如果 A 和 B 其中一个为 true，另一个为 false。当操作符为“&&”时，返回 false；操作符为“||”时，返回 true。
- ❑ 如果 A 和 B 同为 false。当操作符为“&&”时，返回 false；操作符为“||”时，则返回 false。
- ❑ 否则，执行过程产生异常，报错。

注意：在执行过程中，只要表达式结果一旦确定，运算过程就会停止，例如，A && B && C 或者 A and B and C，则一旦计算到 A 为 false，就会立即停止计算，并返回 false。

“||”操作符也同样，例如 A || B || C，则一旦检测到第一个（A、B 或者 C）为 true，则停止计算，直接返回 true。

对“&&”操作符来讲，只要检测到一个 false，则整体为 false；对“||”操作符来说，只要检测到一个 true，则整体返回 true。

18.2.7.2　“非”操作符：!A

执行该操作符之前，会强行把 A 转换成 Boolean 类型，然后再执行“非”或者“反”操作。

- 如果 A 判断为 false，执行“！”操作之后，返回结果 true
- 如果 A 判断为 true，执行完“！”操作之后，返回结果 false
- 否则，执行过程发生异常，报错

18.2.8　Empty操作符

这个“empty”操作符是一个前缀形式的操作符，它使用来判断某个变量是否为 NULL 或者为空，在 EL 中的使用格式如下：${empty user.name}，判断 user 对象 name 属性值是否为 NULL 或者空。“empty”操作符的运算规则如下：

- 如果 A 为 NULL，则返回 true。
- 否则，如果 A 是一个空的字符串，则返回 true。
- 否则，如果 A 是一个空的数组，则返回 true。
- 否则，如果 A 是一个空的 Map 集合类，则返回 true。
- 否则，如果 A 是一个空的 List 集合类，则返回 true。
- 否则，返回 false。

18.2.9　其他操作符

包括的其他操作符还有：条件操作符和括号操作符。

18.2.9.1　条件操作符

EL 中还可以使用以下形式的条件操作符：

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

其中 A 为判断表达式，当 A 为 true 时，返回 B 表达式执行的结果；如果 A 为 false，则返回 C 表达式执行之后的结果。

18.2.9.2　括号操作符

在 EL 中也可以包含括号，例如：

```
${ (a * (b + c) ) }
```

括号主要是使用来改变执行优先权，所有操作符的优先级顺序如下（由上往下，由左往右）：

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

18.3　EL中的隐含对象

为了方便地获得 Web 应用程序中的相关数据，EL 表达式语言定义了一些隐含对象。隐含对象总共有 11 个，这样使得 EL 能够更加方便地获取到数据。

这 11 隐含对象能够很方便地读取到 session、cookie、HttpHeader 以及用户提交表单（param）等，具体请查看如表 18.4 所示。

注意：有很多读者会把 EL 中定义的隐含对象和 JSP 中定义的隐含对象相混淆，其实只有一个对象是它们两者所共有的，即 PageContext 对象。这样 EL 也可以访问到 JSP 中的其他隐含对象，这是因为 PageContext 对象拥有访问 JSP 中所有其他 8 个隐含对象的权限。实际上，这也是将 PageContext 对象包含在 EL 隐含对象中的主要原因。

表 18.4　EL中的隐含对象

类别	隐含对象	描述
JSP	pageContext	当前JSP页面的javax.servlet.jsp.PageContext对象
作用域	pageScope	页面范围内的所有对象的集合
	requestScope	所有请求范围内的对象集合
	sessionScope	所有会话范围内的对象集合
	applicationScope	应用程序范围内的对象变量组成的集合
请求参数	param	所有请求参数字符串组成的集合
	paramValues	所哟作为字符串集合的请求参数
请求头	header	HTTP请求头部，字符串
	headerValues	HTTP请求头部，字符串集合
Cookie	cookie	所有cookie组成的集合
初始化参数	initParam	全部应用程序参数名组成的集合

除了第一个 pageContext 内置对象之后，其余所有 EL 隐含对象都是映射，可以用来查找对应于名称的对象。接下来还是通过一系列实例来讲解这些隐含对象的使用方法：

18.3.1　EL中的pageContext隐含对象实例演示

创建一个 pageContext.jsp 页面，在页面中实现在 EL 中对 pageContext 隐含对象的调用。其源代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>pageContext 隐含对象在 EL 中使用</title></head>
<body>
    <h3>EL 中的 pageContext 隐含对象</h3>
    <hr>
    <table border="1">
        <tr>
            <td align="center">pageContext 隐含对象的方法调用</td>
            <td align="center">结果</td>
      </tr>
      <tr><!-- 取得请求的参数字符串 -->
```

```
            <td>\${pageContext.request.queryString}:</td>
            <td><c:out value="${pageContext.request.queryString}" /></td>
        </tr>
        <tr><!-- 取得请求的 URL，但不包含请求的参数字符串 -->
            <td>\${pageContext.request.requestURL}:</td>
            <td><c:out value="${pageContext.request.requestURL}" /></td>
        </tr>
        <tr><!-- Web application 的名称 -->
            <td>\${pageContext.request.contextPath}:</td>
            <td><c:out value="${pageContext.request.contextPath}" /></td>
        </tr>
        <tr><!-- 取得 HTTP 的方法（GET 或者 POST） -->
            <td>\${pageContext.request.method}:</td>
            <td><c:out value="${pageContext.request.method}" /></td>
        </tr>
        <tr><!-- 取得使用的协议（HTTP/1.1、HTP/1.0） -->
            <td>\${pageContext.request.protocol}:</td>
            <td><c:out value="${pageContext.request.protocol}" /></td>
        </tr>
        <tr><!-- 取得用户名称 -->
            <td>\${pageContext.request.remoteUser}:</td>
            <td><c:out value="${pageContext.request.remoteUser}" /></td>
        </tr>

        <tr><!-- 取得用户的 IP 地址 -->
            <td>\${pageContext.request.remoteAddr}:</td>
            <td><c:out value="${pageContext.request.remoteAddr}" /></td>
        </tr>
        <tr><!-- 判断 session 是否为新的。所谓新的 session，就是刚由 server 产生而 client 尚未使用的 -->
            <td>\${pageContext.session.new}:</td>
            <td><c:out value="${pageContext.session.new}" /></td>
        </tr>
        <tr><!-- 取得 session 的 ID -->
            <td>\${pageContext.session.id}:</td>
            <td><c:out value="${pageContext.session.id}" /></td>
        </tr>
        <tr><!-- 取得主机信息 -->
            <td>\${pageContext.servletContext.serverInfo}:</td>
            <td><c:out value="${pageContext.servletContext.serverInfo}" /></td>
        </tr>
    </table>
</body>
</html>
```

在 IE 浏览器中输入 http:localhost:8080/jstltest/pageContext.jsp?test=test1 地址访问 pageContext.jsp 页面，其运行效果如图 18.3 所示。

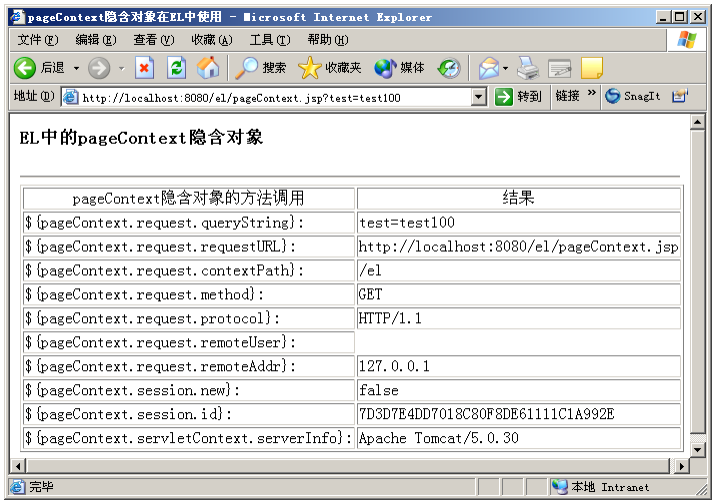

图 18.3　pageContext.jsp 页面效果

18.3.2　EL中header、headerValues、param、paramValues隐含对象实例演示

这四个映射用来获取请求参数和请求头的值。因为 HTTP 协议允许请求参数和请求头具有多个值，所以它们各有一对映射。每对中的第一个映射返回请求参数或头的主要值，通常是恰巧在实际请求中首先指定的那个值。每对中第二个映射允许检索参数或头的所有值。这些映射中的键是参数或头的名称，但这些值是 String 对象的数组，其中的每个元素都是单一参数值或头值。

param 和 paramValues 两隐含对象是取得用户的请求参数，类似于 JSP 中的下列方法：

- request.getParameter(String name)
- request.getParameterValues(String name)

下面创建 paramsubmit.jsp 和 param.jsp 页面来演示这四个隐含变量的使用方法，paramsubmit.jsp 用于提交参数，其代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<html>
<head><title>header,headerValues,param,paramValues 隐含对象在 EL 中使用</title></head>
<body>
      <h3>EL 中的隐含对象：header,headerValues,param,paramValues</h3>
      <hr>
<form action="param.jsp" method="post">
<table width="400" align="center" border="1">
  <tr>
    <td width="60">姓名：</td>
    <td> <input type="text" name="name" width="15" /></td>
  </tr>
  <tr>
    <td>性别：</td>
    <td> <input type="radio" name="sex" value="male" checked />男
          <input type="radio" name="sex" value="female" />女</td>
  </tr>
  <tr>
    <td>专业：</td>
```

```
    <td> <select name="specialty">
            <option value="计算机科学" selected>计算机科学</option>
             <option value="通信">通信</option>
             <option value="管理科学与工程">管理科学与工程</option>
             </select></td>
  </tr>
  <tr>
    <td>爱好：</td>
    <td> <input type="checkbox" name="habit" value="电影"/>电影
          <input type="checkbox" name="habit" value="读书"/>读书
           <input type="checkbox" name="habit" value="羽毛球"/>羽毛球
           <input type="checkbox" name="habit" value="篮球"/>篮球
           <input type="checkbox" name="habit" value="旅游" />旅游</td>
  </tr>
  <tr>
    <td colspan="2" align="center"><input type="submit" value="提交"/> | <input type="reset" value="重置
"/></td>
  </tr>
</table>
</form>
</body>
</html>
```

此页面的运行效果如图 18.4 所示。把信息填好后，单击“提交”按钮，页面定向到 param.jsp，此页面的源代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ 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>header,headerValues,param,paramValues 隐含对象在 EL 中使用</title></head>
<body>
      <h3>EL 中的隐含对象：header,headerValues,param,paramValues</h3>
      <hr>
<fmt:requestEncoding value="GBK" />
HTTP 连接头部的 host 值: ${header["host"]} <br>
HTTP 头部的 accept 值: ${header.accept} <br>
HTTP 头部的 user-agent 值: ${header["user-agent"]} <br>
<hr width="50%" align="left">
姓名：<c:out value="${param.name}" /> <br>
性别：<c:out value="${param.sex}" /> <br>
专业：<c:out value="${param['specialty']}" /> <br>
兴趣：<c:out value="${paramValues.habit[0]}" />
        <c:out value="${paramValues.habit[1]}" />
         <c:out value="${paramValues.habit[2]}" />
         <c:out value="${paramValues.habit[3]}" />
         <c:out value="${paramValues.habit[4]}" />
</body>
</html>
```

程序说明：在程序中需要加上< fmt:requestEncoding value="GBK" />代码，不然上一页面传递过程的

参数是乱码。

此页面运行效果如图 18.5 所示。

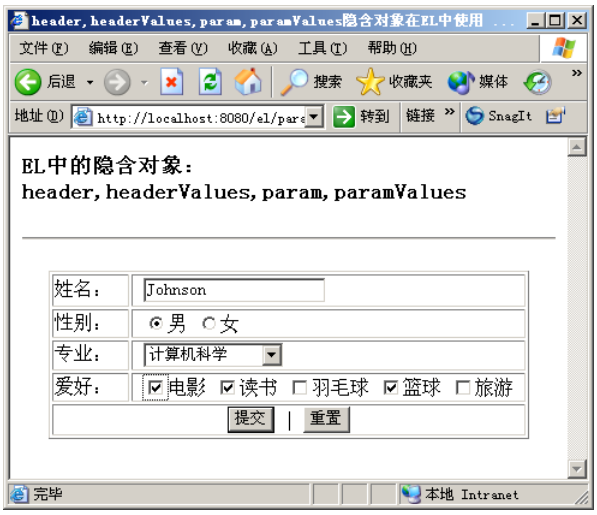

图 18.4　提交页面

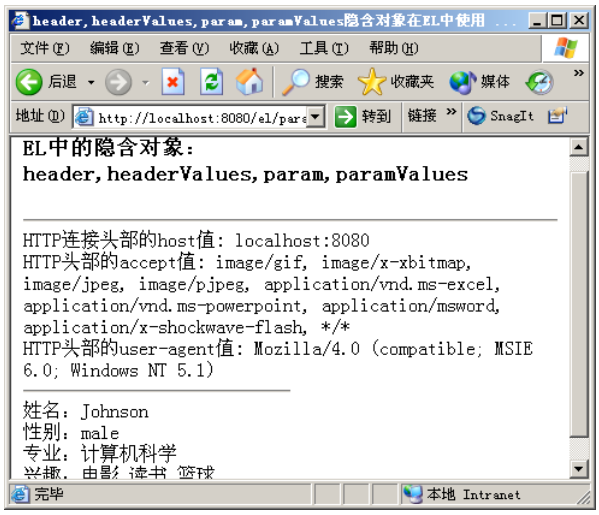

图 18.5　param.jsp 页面效果

18.3.3　EL中pageScope、requestScope、sessionScope、applicationScope隐含对象实例演示

这四个隐含对象表示先前讨论的各种属性作用域。可以用它们来查找特定作用域中的标识符，而不用依赖于 EL 在默认情况下使用的顺序（首先 page，然后 request、session，再 application）查找过程。

它们和 JSP 中的 page、request、session 以及 application 隐含对象非常相似，但是有一点要注意的，这里的四个隐含对象只能用来取得范围属性值，而不能取得其他相关信息。

下面创建一个 scope.jsp 文件。首先使用<c:set>标签在不同的范围内（page、request、session 和 application）设置相同的 name 变量，然后调用相应的 pageScope、requestScope、sessionScope 以及 applicationScope 隐含对象来取得 name 变量值。

scope.jsp 文件的源代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head><title>pageScope、requestScope、sessionScope 以及 applicationScope 隐含对象在 EL 中使用
</title></head>
<body>
      <h4>EL 中的 pageScope、requestScope、sessionScope 以及 applicationScope 隐含对象</h4>
      <hr>
<c:set var="name" value="pageValue" scope="page" />
<c:set var="name" value="requestValue" scope="request" />
<c:set var="name" value="sessionValue" scope="session" />
<c:set var="name" value="applicationValue" scope="application" />

\${name}: <c:out value="${name}" /> <br>
\${pageScope.name}: <c:out value="${pageScope.name}" /> <br>
\${requestScope.name}: <c:out value="${requestScope.name}" /> <br>
```

```
\${sessionScope.name}: <c:out value="${sessionScope.name}" /> <br>
\${applicationScope.name}: <c:out value="${applicationScope.name}" /> <br>
</body>
</html>
```

程序说明：由图 18.6 所示，可知当不调用相应的隐含对象来取得 name 变量时，容器会首先查到 page 范围内的 name 变量。

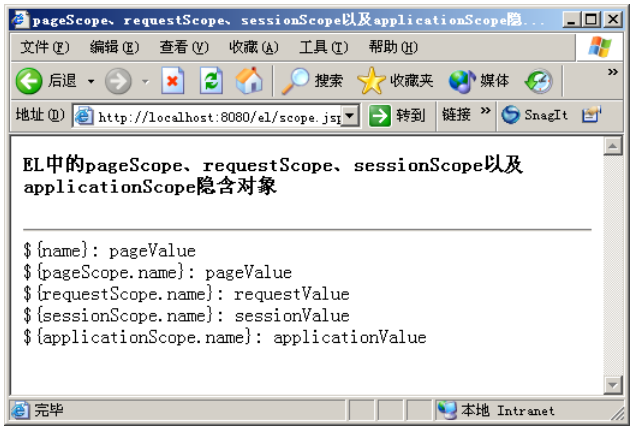

图 18.6　scope.jsp 页面

18.3.4　cookie和initParam隐含对象

cookie 隐式对象提供了对由请求设置的 cookie 名称的访问。这个对象将所有与请求相关联的 cookie 名称映射到表示那些 cookie 特性的 Cookie 对象。

JSTL 标准标签并没有提供设置 cookie 的动作，因此这个动作通常都是开发者必须要去做的事情，而不是交给前端。如已经在 cookie 中设置了一个 name 的值，则可以通过${cookie.name}来取得它的值。

最后一个 EL 隐式对象 initParam 也是一个映射，它储存与 Web 应用程序相关联的所有上下文的初始化参数的名称和值。初始化参数是通过 web.xml 部署描述符文件指定的，该文件位于应用程序的 WEB-INF 目录中。例如在 web.xml 文件中设定如下：

```
<context-param>
   <param-name>username</param-name>
   <param-value>Johnson</param-value>
</content-param>
```

那么在程序当中就可以直接使用${initParam.username}来取得 username 参数的值“Johnson”。

18.4　EL中调用自定义函数

EL 中可以对自定义函数的方法进行调用，下面还是通过一个实例进行示范，也体会一下 JSTL 和 EL 的完美配合。首先创建一个函数类，JSTL 对函数类没有任何要求，只要求方法类是公开的，方法必须是静态的、公用的方法。下面是这个函数类 EmailFunction.java 实现的代码，把它放在包 cn.el 中：

```
package cn.com.zzb;
public class EmailFunction {
 /**
  * 转换 EMAIL 地址为链接的形式
```

```
   * @param email
   * @return
   */
  public static String emailLink(String email){
   StringBuffer sb = new StringBuffer();
   sb.append("<a href=\"mailto:");
   sb.append(email);
   sb.append("\">");
   sb.append(email);
   sb.append("</a>");
   return sb.toString();
  }
}
```

程序说明：上面程序非常简单，emailLink(String email)方法实现把 email 地址转换成链接形式。

接下来我们必须通知 JSTL 怎么来使用这个函数，跟标签库一样，我们必须编写一个 tld 文件（查看第十六章的内容），姑且把文件名叫做 email.tld。该 tld 文件存放在 WEB-INF/tlds 目录（或者 WEB-INF 目录的其他子目录下），该文件中包含对该函数的说明，文件内容如下：

```
<?xml version="1.0" encoding="GB2312" ?>
<taglib 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
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
   version="2.0">

   <description>Demo</description>
   <display-name>Email functions</display-name>
   <tlib-version>1.1</tlib-version>
   <short-name>mail</short-name>
   <uri>http://www.jstl.com/dev/jsp/jstl/mail</uri>
   <function>
      <description>
         用于将电子邮件转成链接形式
      </description>
      <name>emailLink</name>
      <function-class>cn.el.EmailFunction</function-class>
      <function-signature>java.lang.String emailLink(java.lang.String)</function-signature>
      <example>
         ${mail:emailLink("javayou@gmail.com")}
      </example>
   </function>

</taglib>
```

接下来就是要编写一个测试页面，这个页面的代码如下：

```
<%@ page contentType="text/html; charset=GBK" %>
<%@ taglib prefix="em" uri="http://www.javayou.com/dev/jsp/jstl/mail" %>
<%@ page isELIgnored="false" %>
Click ${em:emailLink("zzb@gmail.com")} to feedback.
```

打开 IE 浏览器，并运行该页面，显示的结果如下所示：

```
Click zzb@gmail.com to feedback.
```

18.5　本章小结

EL 表达式语言起初只是作为标准标签库的一部分，也只能在 JSTL 中使用，但是随着 JSP2.0 版本的发布，EL 已经能够在 JSP 页面中非常方便地访问数据。EL 和 JSTL 标准标签库的配合使用可以使得 Web 开发者不再需要编写包含脚本或者 JavaBean 而创建 Web 应用程序。从而可以达到面向文档的开发方式，使得程序更加的简洁、更易于维护。另外 EL 还很好地支持了自定义函数的调用。