第5章 EL表达式与JSTL标签库

B/S架构的Web程序开发与C/S模式不同，需要解决两大难题。其一是如何把客户端浏览器发送过来的请求提交给服务器，对此一般使用网页中的Form表单、框架中的数据封装或参数绑定等；其二则是如何把服务器端处理所得的结果数据显示到客户端的浏览器上，对此一般采用JSP的内置对象、JSP表达式或表达式语言来实现。

在Web应用程序中，视图层的开发技术除了HTML、CSS、JavaScript和JSP之外，还有EL(表达式语言)、JSTL(JSP标准标签库)和Ajax技术等。


视频讲解

5.1 EL表达式

表达式语言(Expression Language，EL)是JSP标准的一部分，是JSP 2.0新增加的特性之一。EL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，后来成了JSP标准的一部分，如今EL已经成为一项成熟、标准的技术。EL在JSP中的引入可以大幅度地减少Java代码的编写。由于EL是JSP 2.0新增的功能，所以只有支持Servlet 2.4/JSP 2.0的容器，才能在JSP网页中直接使用EL，Tomcat 9.0中可以直接使用EL。

5.1.1 EL的语法

EL并不是一种通用的编程语言，它只是一种数据访问语言。网页作者通过它可以很方便地在JSP页面中访问应用程序数据，无须使用小脚本(<%…%>)或JSP请求时表达式(<%=…%>)。作为一种数据访问语言，EL具有自己的标识符、运算符、语法和保留字等。

1. 语法形式

在JSP 2.0的页面中，表达式语言的使用形式为：${expression}。

表达式以$开头，后面跟一对大括号{}，括号中的expression是一个合法的EL表达式。该结构可以出现在JSP页面的body区文本中，也可以出现在JSP标签的属性值里，只要属性允许常规的表达式即可。下面通过一个例子，来直观感受一下EL表达式的优越性。

【例5-1】 使用EL表达式读取域对象中的数据。

(1) 在Eclipse中新建一个名为0501-SimpleEL的项目。

(2) 新建一个User类，放置在cn.pju.entity包下，详细代码如下。

```
package cn.pju.entity;

public class User {
```

```
    private String uName;
    private String uGender;

    public User() {
        super();
    }
    public User(String uName, String uGender) {
        super();
        this.uName = uName;
        this.uGender = uGender;
    }
    public String getuName() {
        return uName;
    }
    public void setuName(String uName) {
        this.uName = uName;
    }
    public String getuGender() {
        return uGender;
    }
    public void setuGender(String uGender) {
        this.uGender = uGender;
    }
}
```

（3）新建一个名为 SimpleELServlet 的 Servlet，继承自 HttpServlet 父类，放置在 cn.pju.servlet 包下，代码如下。

```
package cn.pju.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import cn.pju.entity.User;

@WebServlet("/SimpleELServlet")
public class SimpleELServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public SimpleELServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        User user = new User();
        user.setuName("张慧");
        user.setuGender("女");
```

```
            request.setAttribute("user", user);
            request.getSession().setAttribute("password", "123456");
            request.getRequestDispatcher("/index.jsp").forward(request, response);
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
            doGet(request, response);
        }

    }
```

(4) 在 WebContent 目录下,新建 index.jsp 文件,代码如下。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="cn.pju.entity.User" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>EL 的基本用法</title>
</head>
<body>
    使用 Java 代码读取 Request 域中的对象<br>
    <%=((User)request.getAttribute("user")).getuName() %><br>
    <%=((User)request.getAttribute("user")).getuGender() %><br>
    <%=session.getAttribute("password").toString().trim() %><br>
    <hr>
    使用 EL 表达式读取<br>
    ${requestScope.user.uName}<br>
    ${requestScope.user.uGender}
    ${sessionScope.password}
</body>
</html>
```

以上程序的主要工作是在名为 SimpleELServlet 的 Servlet 中定义了一个 User 类的实例对象 user,并对 user 进行了初始化,写入了姓名和性别两个属性。然后将 user 对象写入 request 域对象,向 session 域中写入名为 password 的一个字符串值,然后转发到 index.jsp 页面。

在客户端视图层的 index.jsp 文件中,使用两种方式分别读取 request 域中的 user 对象和 session 域中的 password 字符串,普通的 Java 代码为:

```
<%=((User)request.getAttribute("user")).getuName() %>
<%=((User)request.getAttribute("user")).getuGender() %>
```

而使用 EL 表达式则十分简洁:

```
${requestScope.user.uName}<br>
${requestScope.user.uGender}<br>
${sessionScope.password}
```

其中的 requestScope 和 sessionScope 是可以省略的，如果不指定搜索范围，则依次在 page、request、session、application 范围中查找，所以上述代码还可以简化为

```
${user.uName}<br>
${user.uGender}<br>
${password}
```

显然，使用 EL 编写输出域对象数据代码时，代码量明显减少，条理清晰，工作效率高。

(5) 将项目 0501-SimpleEL 部署到 Tomcat 服务器上，启动服务器，在浏览器地址栏中输入网址 http://localhost:8080/0501-SimpleEL/SimpleELServlet 进行测试，结果如图 5-1 所示。

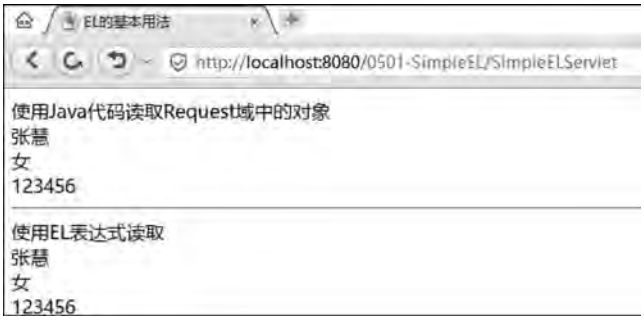

图 5-1　使用 EL 表达式读取域对象中数据

下面的代码是在 JSP 模板文本中使用 EL 表达式读取客户姓名和性别的例子。

```
<ul>
    <li>客户姓名：${customer.cName}</li>
    <li>客户性别：${customer.cGender}</li>
</ul>
```

下面的代码是在 JSP 标准动作的属性中使用 EL 表达式的例子。

```
<jsp:include page="${expression1}" />
<c:out value="${expression2}" />
```

2. EL 中的标识符

在 EL 表达式中，经常需要使用一些符号来标记一些对象的名称，如变量名、自定义函数名等，这些符号称作标识符。EL 表达式中标识符的命名规则如下。

(1) 由数字、字母(大小写均可)和下画线组成。

(2) 数字不能作为开头。

(3) 不能是 EL 中的保留字或隐式对象。

3. EL 中的保留字

保留字，就是编程语言自己使用的一些标识符，具有特定的含义，不能被定义为用户级的标识符。EL 中常见的保留字有 and、div、empty、eq、false、ge、gt、instanceof、le、lt、mod、ne、not、null、or、true 等。

4. EL 的功能

引入 EL 的目的就是要简化页面的表示逻辑，其主要功能如下。

(1) 简化运算。表达式语言提供了一组简单有效的运算符，通过这些运算符可以快速

完成算术、关系、逻辑、条件或空值检验等运算。

(2) 便捷访问作用域变量。在 Web 程序开发中,可以使用 setAttribute()函数将变量写入到 PageContext、HttpServletRequest、HTTPSession 或 ServletContext 作用域中,使用 EL 可以简单快捷地进行读取。

(3) 便捷访问 JavaBeans 对象。在 JSP 页面中要访问一个 JavaBean 对象 user 的 userName 属性,需要使用语句<jsp:getProperty name="user" property="userName">,而使用 EL 表达式,可以简化表示为 ${user.userName}。

(4) 简化访问集合元素。集合主要包括数组 Array、List 对象和 Map 对象等,对这些对象的元素进行访问可以使用 ${var[indexOrKey]}直接完成。

(5) 对请求参数、Cookie 和其他请求数据进行简单访问。例如,要访问 Accept 请求头,可以使用 header 隐含变量来实现,代码如下。

```
${header.Accept}或${header["Accept"]}
```

(6) 调用 Java 外部函数。EL 中不能定义和使用变量,也不能调用对象的方法,但可以通过标签的形式使用 Java 语言定义的函数。

5. EL 与 JSP 表达式的区别

(1) 使用 EL 表达式和 JSP 表达式都可以向表示层页面输出数据,但二者表示形式不同。JSP 表达式的使用格式为<%=expression%>,其中,expression 为合法的 Java 表达式,它属于脚本语言代码; EL 表达式的格式为 ${expression},这里的 expression 为符合 EL 规范的表达式,并且不需要包含在标签里。

(2) JSP 表达式中可以使用由 Java 脚本声明的变量,在 EL 表达式中却不能使用由脚本语句声明的变量。例如,有以下 JSP 脚本:

```
<%! int x = 18; %>
x 的值为: <% =x%>
```

则运行该脚本后的显示结果为 18。如果把以上代码修改为:

```
<%! int x = 18; %>
x 的值为: ${x}
```

则运行结果为空值(EL 的 empty 运算符测试结果为 true)。

在 EL 中不能定义变量,也不能使用脚本中声明的变量,但可以通过 EL 访问请求参数、作用域变量、JavaBeans 以及 EL 隐含变量等。

(3) EL 中不允许调用对象的方法,但是 JSP 表达式可以。比如 ${pageContext.request.getMethod()}的用法是错误的,但是可以使用脚本表达式<%=request.getMethod()%>来正常调用。

5.1.2 EL 的运算符

EL 表达式是由标识符和运算符构成的式子,EL 本身定义了一些用来存取、处理或比较的操作运算符。

1. 存取运算符

在 EL 中,对数据的存取通过“.”或“[]”来实现,有时也把二者称作属性运算符和集合

元素运算符。其格式为：

```
${objName.property}  或  ${objName["property"]  或  ${objName[property]}
```

说明：

(1) “.”运算符主要用于访问对象的属性。

(2) “[]”运算符主要用来访问数组、列表或其他集合对象的属性。

(3) “.”和“[]”在访问对象属性时可以通用，但也有如下区别。

① 当存取的属性名包含特殊字符(如.或-等非数字和字母符号)时，就必须使用“[]”运算符，如${customer["c-name"]}。

② “[]”中可以是变量，“.”后只能是常量，如${user[gender]}、${user.gender}、${user["gender"]}，其中后两者等价。

2. 算术运算符

算术运算符如表 5-1 所示。

表 5-1　算术运算符

算术运算符	功　能	举　例	结果	备　注
+	加	${3+2}	5	
−	减或负号	${3−2}	1	
*	乘	${1.5e2 * 2}	300	$1.5\times10^2\times2$
/(或 div)	除	${10/4}或${10 div 4}	2.5	除法运算的商为小数
%(或 mod)	取余(取模)	${10%4}或${10 mod 4}	2	两个操作数必须是整数

3. 关系运算符

关系运算符(比较运算符)如表 5-2 所示。

表 5-2　关系运算符(比例运算符)

关系运算符	功　能	举　例	结果	备　注
>(或 gt)	大于	${10>4}或${10 gt 4}	true	greater than
>=(或 ge)	大于等于	${10>=4}或${10 ge 4}	true	greater than or equal
<(或 lt)	小于	${10<4}或${10 lt 4}	false	less than
<=(或 le)	小于等于	${10<=4}或${10 le 4}	false	less than or equal
==(或 eq)	等于	${10==4}或${10 eq 4}	false	equal
!=(或 ne)	不等于	${"a"!="xy"}或${"a" ne "xy"}	true	not equal

4. 逻辑运算符

逻辑运算符如表 5-3 所示。

表 5-3　逻辑运算符

逻辑运算符	功　能	举　例	结果	备　注
!(或 not)	逻辑非	${!true}或${not true}	false	
&&(或 and)	逻辑与	${true&&false}或${true and false}	false	注意逻辑与短路
||(或 or)	逻辑或	${true\|\|false}或${true or false}	true	注意逻辑或短路

5. 条件运算符

EL 的条件运算符类似于 C 语言中的问号表达式，其语法是：booleanExp? ex1:ex2。

系统首先计算 booleanExp 表达式的值，如果结果为 true，则将表达式 ex1 的值作为整个条件表达式的值返回；若 booleanExp 表达式的结果为 false，则返回表达式 ex2 的结果。

6. empty 运算符

EL 表达式中的 empty 运算符用于判断某个对象是否为 null 或空字符串，结果为布尔类，其语法格式为：${empty var}。说明如下。

(1) 若 var 变量不存在(即没有定义)，例如表达式 ${empty name}，如果不存在 name 变量则返回 true。

(2) 若 var 变量的值为 null，例如表达式 ${empty customer.name}，如果 customer.name 没有赋值，则其值为 null，就返回 true。

(3) 若 var 变量引用集合(Set、Map 和 List)类型对象，并且在集合对象中不包含任何元素时，则返回值为 true。例如，如果表达式 ${empty list}中 list 集合中没有任何元素，就返回 true。

empty 运算符的返回值如表 5-4 所示。

表 5-4 empty 运算符的返回值

var 值	${empty var}返回值	var 值	${empty var}返回值
null	true	空 Map	true
""(空 String)	true	空 Collection	true
空 Array	true	其他	false

7. ()运算符

在 EL 表达式中可以通过括号()来调整运算符的先后执行顺序。表 5-5 给出了 EL 表达式中常见运算符的优先级顺序。

表 5-5 运算符的优先级

优先级	运 算 符	优先级	运 算 符
1	[]	6	< > <= >= lt gt le ge
2	()	7	== != eq ne
3	- not ! empty	8	&& and
4	* / div % mod	9	|| or
5	+ -	10	?:

8. EL 中的自动类型转换

EL 提供自动类型转换功能，能够按照一定规则将操作数或结果转换成指定的类型，表 5-6 列举的是自动类型转换的实例。

表 5-6　EL 自动类型转换的实例

EL 表达式	结　果	说　明
${true}	"true"	boolean 转 String
${false}	"false"	boolean 转 String
${null}	""	null 转 String(空串)
${null + 0}	0	null 转 Number
${"123.5" + 0}	123.5	String 转 Number
${"1.2E3" + 0.5}	1200.5	String 转 Number

5.1.3　EL 的隐含对象

在学习 JSP 技术时,共提到了 JSP 的 9 大隐含对象(Implicit Object),而 EL 本身也有自己的隐含对象。EL 隐含对象总共有 11 个,如表 5-7 所示。

表 5-7　EL 中的隐含对象

隐含对象	类　型	说　明
pageContext	javax.servlet.ServletContext	对应于 JSP 页面中的 pageContext 对象
pageScope	java.util.Map	取得 page 域中保存属性的 Map 对象
requestScope	java.util.Map	取得 request 域中保存属性的 Map 对象
sessionScope	java.util.Map	取得 session 域中保存属性的 Map 对象
applicationScope	java.util.Map	取得 application 域中保存属性的 Map 对象
param	java.util.Map	表示一个保存了所有请求参数的 Map 对象
paramValues	java.util.Map	表示一个保存了请求参数字符串数组的 Map 对象
header	java.util.Map	表示一个保存了所有 HTTP 请求头字段的 Map 对象
headerValues	java.util.Map	包含请求头字符串数组的 Map 对象
cookie	java.util.Map	取得使用者的 cookie 值,cookie 的类型为 Map
initParam	java.util.Map	一个保存了所有 Web 应用初始化参数的 Map 对象

(1) ${pageContext}获取到 pageContext 对象,它不是在四个域里面去找,而是先在自己定义的对象中找,如果找到了就取出来。

(2) ${pageScope}得到的是 page 域(pageContext)中保存数据的 Map 集合。也就是指定在 page 域中查找。

(3) ${requestScope}、${sessionScope}、${applicationScope} 和上面的 pageScope 一样,都是在特定的域中检索数据。

(4) ${param} 获取存在 request 中请求参数的 Map,常用在数据回显上。

(5) ${paramValues}获取存在 request 中请求参数名相同的值的 String[]数组。

(6) ${header}获取 HTTP 请求头的 Map 对象。

(7) ${headValues}获取 HTTP 请求头值中的 Map 对象。

(8) ${cookie}获取所有 cookie 的 Map 对象。

(9) ${initParam}获取保存所有 Web 应用初始化参数的 Map 对象。

下面对这 11 种对象做详细讲解。

1. pageContext 对象

pageContext 是一个特殊的对象，可以获取整个页面的上下文环境，通过 pageContext 可以获取其他 10 个隐式对象。尤其是使用 EL 表达式中的 pageContext 对象可以轻松地获取到 request、response、session、out、servletContext 和 servletConfig 对象中的属性。需要注意的是，不要将 EL 表达式中的隐式对象与 JSP 中的隐式对象混淆。只有 pageContext 对象是二者所共有的，其他隐式对象则毫不相关。pageContext 对象常用表达式如表 5-8 所示。

表 5-8　pageContext 对象常用表达式

表　达　式	说　　明
${pageContext.request.queryString}	取得请求的参数字符串
${pageContext.request.requestURL}	取得请求的 URL，但不包括请求的参数字符串
${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，即刚由 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
${pageContext.response.contentType}	获取 content-type 响应头
${pageContext.servletConfig.servletName}	获取 Servlet 的注册名你

上述 EL 是通过成员访问运算符访问对象的属性。在 EL 表达式中不允许调用对象的方法，所以下面的使用是错误的。

```
${pageContext.request.getMethod()}
```

但是，在 Java 代码中却可以正常使用函数，如下面的语句。

```
<% = request.getMethod() %>
```

2. Web 域相关对象

pageScope、requestScope、sessionScope 和 applicationScope 是用于获取指定域中数据的隐式对象。EL 中的 4 大域对象与 JSP 中域对象的对应关系如表 5-9 所示。

表 5-9　EL 域对象与 JSP 域对象的对应关系

EL 隐含对象	JSP 对应域对象	说　　明
pageScope	pageContext	页面作用域对象，只在当前页面内有效
requestScope	request	在用户的请求和转发的请求内有效
sessionScope	session	在一个用户的会话范围内有效
applicationScope	application	在整个 Web 应用程序内都有效

在 Web 开发中，PageContext、HttpServletRequest、HttpSession 和 ServletContext 这 4 个对象之所以可以存储数据，是因为它们内部都定义了一个 Map 集合，人们习惯把这些

Map 集合称为域，这些 Map 集合所在的对象称为域对象。这些 Map 集合是有一定作用范围的。例如，HttpServletRequest 对象存储的数据只在当前请求中才可以获取到。在 EL 表达式中，为了获取指定域中的数据，系统提供了 pageScope、requestScope、sessionScope 和 applicationScope 4 个隐含对象。需要注意的是，EL 表达式只能在这 4 个作用域中获取数据。

```
${pageScope.userAddress}
${requestScope.userAddress}
${sessionScope.userAddress}
${applicationScope.userAddress}
```

为了更好地理解 EL 读取域对象中的数据，下面通过两个例子来加深理解。

【例 5-2】 使用 EL 表达式读取 4 种域对象中的数据(通过 JSP 写入)。

(1) 在 Eclipse 中新建一个名为 0502-FieldsData 的项目。

(2) 在 WebContent 目录下新建两个 JSP 文件。

① inData.jsp 代码如下。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>向域对象写入数据</title>
</head>
<body>
  <%
    pageContext.setAttribute("pageData", "OOP");
    request.setAttribute("requestData", "Java Web");
    session.setAttribute("sessionData", "Java EE");
    application.setAttribute("appData", "SSM");
    request.getRequestDispatcher("outData.jsp").forward(request, response);
  %>
</body>
</html>
```

② outData.jsp 代码如下。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>读取域对象中的数据</title>
</head>
<body>
```

```
    ${pageScope.pageData}<br>
    ${requestScope.requestData}<br>
    ${sessionScope.sessionData}<br>
    ${applicationScope.appData}
</body>
</html>
```

在 Tomcat 服务器中部署项目，运行 inData.jsp，测试结果如图 5-2 所示。

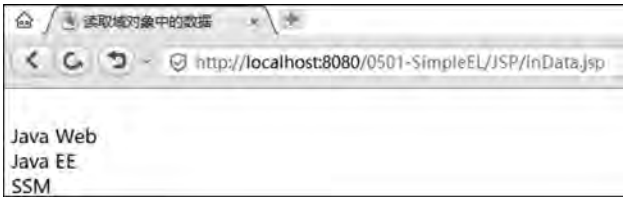

图 5-2　读取域对象数据结果

由图 5-2 所示结果可知，requestScope、sessionScope 和 applicationScope 域对象中的数据都能正常读取，但 pageScope 中的数据无法显示。分析 inData.jsp 页面的代码发现，内含一个页面跳转语句 request.getRequestDispatcher("outData.jsp").forward(request, response)，很显然，在访问 inData.jsp 时，系统先向 4 个域对象写入数据，然后再跳转到一个新的页面 outData.jsp。pageScope 对应的范围是单个页面，由于写入数据是在 inData.jsp 页面完成的，而跳转到 outData.jsp 页面后，由于页面发生了变化，原先写入 pageScope 中的数据 pageData 也会随之消失，所以跳转到 outData.jsp 之后无法正常显示。可以修改 outData.jsp 的代码，重新写入 pageScope 范围内的数据后再进行读取，代码如下。

③ 修改后的 outData.jsp 代码如下。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>读取域对象中的数据</title>
</head>
<body>
  <%
      pageContext.setAttribute("pageData", "面向对象程序设计");
   %>
   ${pageScope.pageData }<br>
   ${requestScope.requestData}<br>
   ${sessionScope.sessionData}<br>
   ${applicationScope.appData}
</body>
</html>
```

代码修改后，重新运行 inData.jsp，显示结果如图 5-3 所示。需要说明的是，使用 EL 表达式获取某个域对象中的属性时，也可以不使用这些隐式对象来指定查找域，而是直接引用

域中的属性名称即可，系统会依次在 page、request、session、application 4 个域范围内搜索对应的属性变量。例如，表达式 ${userAddress} 就是按顺序依次在 page、request、session、application 这 4 个作用域内查找名为 userAddress 的属性变量，获取其值。采用此种方式访问域对象中的数据时，应避免在不同域范围内放入同名变量。

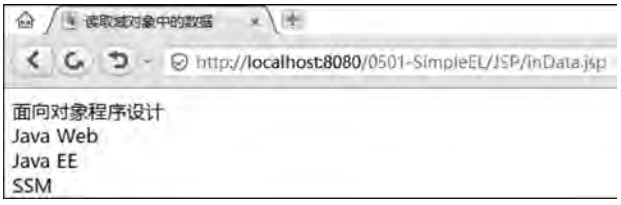

图 5-3　修改后的显示结果

除了使用 JSP 向域中写入数据，还可以在 Servlet 中调用 setAttribute()函数将变量的值存储到某个作用域对象(HttpServletRequest、HttpSession 或 ServletContext)上，然后使用 RequestDispatcher 对象的 forward()将请求转发到 JSP 页面，在 JSP 页面中调用隐含变量的 getAttribute()函数或 EL 语句返回作用域变量的值。

【例 5-3】 使用 EL 表达式读取 4 种域对象中的数据(通过 Servlet 写入)。

(1) 在 Eclipse 中新建一个名为 0503-FieldsDataServlet 的项目。

(2) 新建 cn.pju.servlet 包并在包内新建一个名为 VarIOServlet 的 Servlet 类。

(3) 在 WebContent 目录下新建 vars.jsp 文件。

① VarIOServlet.java 源程序。

```
package cn.pju.servlet;

import java.io.IOException;
import java.util.Date;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/VarIOServlet")
public class VarIOServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public VarIOServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
```

```
        response.setCharacterEncoding("UTF-8");
        //向域中写入数据
        //向 request 域写入字符串数据
        request.setAttribute("requestVar", "请求域数据");
        //向 session 域写入整型数据
        HttpSession session = request.getSession();
        session.setAttribute("sessionVar", new Integer(88));
        //向 application 域写入日期型数据
        ServletContext application = request.getServletContext();
        application.setAttribute("applicationVar", new Date());
        //向 3 个域写入同名数据
        request.setAttribute("varTest", "请求作用域");
        session.setAttribute("varTest", "会话作用域");
        application.setAttribute("varTest", "应用作用域");
        //请求转发到 JSP 页面,在 JSP 页面通过 EL 读取写入的域对象数据
        RequestDispatcher rd = request.getRequestDispatcher("/vars.jsp");
        rd.forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request, response);
    }

}
```

② vars.jsp 源代码。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>访问域对象数据</title>
</head>
<body>
    <h2>通过 EL 访问域中数据</h2>
    request 请求域中数据: ${requestScope.requestVar}<br>
    session 会话域中数据: ${sessionScope.sessionVar}<br>
    application 域中数据: ${applicationScope.applicationVar}<br>
    访问域中同名变量: ${varTest}
</body>
</html>
```

在 Eclipse 中右击选中 VarIOServlet.java 文件,选择 Run as→Run on Server 弹出式菜单,调试运行该 Servlet,得到如图 5-4 所示的运行结果。

3. param 和 paramValues 对象

param 和 parmValues 是用于获取请求参数的隐式对象。

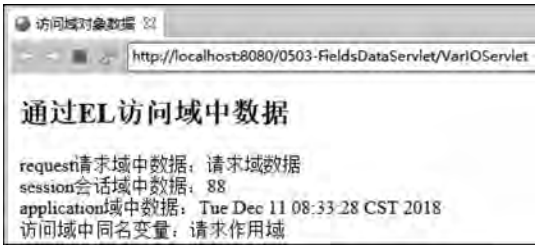

图 5-4　访问域对象数据

在 JSP 页面中，经常需要获取客户端传递的请求参数，例如，在超链接< a href＝"test.jsp? m＝5&n＝8">测试中使用 GET 方式传递的参数 m 和 n。EL 表达式提供了 param 和 paraValues 两个隐式对象，专门用于从 ServletRequest 对象中获取客户端访问 JSP 页面时传递的请求参数。

param 对象用于获取请求参数的某个值，它是 Map 类型，与 JSP 中 request.getParameter(String name)方法相同，在使用 EL 获取参数时，如果参数不存在，则返回空字符串，而不是 null。param 对象的语法格式为：

```
${param.var}
```

param 适用于仅获取一个参数值的情况，如果一个请求参数有多个值，可以使用 paramValues 对象来进行获取，该对象获取到的值列表会存储到一个数组中(数组下标默认从 0 开始)，然后再通过迭代方法获取数组中的每一个值。例如，要获取某个请求参数对应数组中的第一个值，可以使用如下 3 种代码中的任意一个。

```
${paramValues.name[0]}
${paramValues.name["0"]}
${paramValues.name['0']}
```

下面通过一个例子来理解一下 param 和 paramValues 对象。

【例 5-4】 param 和 paramValues 对象的使用。

(1) 在 Eclipse 中新建一个名为 0504-ParamsTest 的项目。

(2) 在 WebContent 目录下新建名为 params.jsp 的 JSP 文件。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>param、paramValues 测试</title>
</head>
<body>
    <center>
        <form action="${pageContext.request.contextPath}/params.jsp" method="post">
            加数 1: <input type="text" name="num" value="${paramValues.num[0]}"><br>
            加数 2: <input type="text" name="num" value="${paramValues.num[1]}"><br>
```

```
            加数 3: < input type = "text" name = "addnum" value = " $ {param.addnum }"> < br >
            num[0]: $ {paramValues.num[0] }< br >
            num[1]: $ {paramValues.num[1] }< br >
            addnum: $ {param.addnum }< br >
            sum: $ {paramValues.num[0] + paramValues.num[1] + param.addnum}< br >
            < input type = "submit" value = "提交" />   
            < input type = "reset" value = "重置" />
        </form >
    </center >
</body >
</html >
```

在 Tomcat 服务器上部署 Web 项目后直接运行测试 params.jsp，输入三个加数值单击“提交”按钮，在下方会显示通过 param 和 paramValues 对象读取的数据，结果如图 5-5 所示。

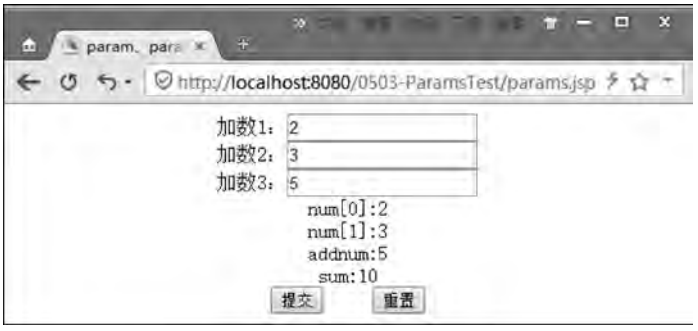

图 5-5　param、paramValues 对象测试

分析网页代码，加数 1 和加数 2 对应的文本输入框 name 属性相同，都是 num，所以通过 EL 进行获取的时候使用的是 paramValues 对象，分别使用下标 0、1 对应读取；加数 3 对应的文本框 name 值为 addnum，在网页中的属性值唯一，可以直接使用 param 对象进行读取。

需要注意的是，如果使用 $ {param.num} 去读取数据，由于加数 1 和加数 2 对应的文本框 name 都是 num，返回的只有第一个值，读者可自行测试。

4. cookie 对象

cookie 是用于获取 Cookie 信息的隐含对象。

在 Web 项目开发中，经常需要获取客户端浏览器的 Cookie 信息。在 EL 表达式中，提供了 Cookie 隐含对象，该对象是一个代表所有 Cookie 信息的 Map(name-value 值对)集合，Map 集合中元素的键为各个 Cookie 的名称 name，值则为对应的 Cookie 对象 value，具体示例如下。

【例 5-5】 使用 EL 表达式读取 cookie 中的数据。

(1) 在 Eclipse 中新建一个名为 0505-CookieDataServlet 的项目。

(2) 新建 cn.pju.servlet 包并在包内新建一个名为 CookieVarServlet 的 Servlet 类。

(3) 在 WebContent 目录下新建 CookieVars.jsp 文件。

① CookieVarServlet.java 源代码。

```
package cn.pju.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/CookieVarServlet")
public class CookieVarServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public CookieVarServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        //在 Servlet 中向客户端发送一个 Cookie,该 cookie 的 name 是"userType",value 是"admin"
        Cookie cookie = new Cookie("userType", "admin");
        response.addCookie(cookie);
        request.getRequestDispatcher("/CookieVars.jsp").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request, response);
    }

}
```

② CookieVars.jsp 源代码。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>读取 Cookie 中的数据</title>
</head>
<body>
    <h2>使用 Java 代码读取</h2>
    <%
        Cookie[] cookies = request.getCookies();
```

```
            for(int i = 0; i < cookies.length; i++){
                out.println(cookies[i].getName() + " === ");
                out.println(cookies[i].getValue() + "< br >");
            }
        % >
        < hr >
        < h2 >使用 EL 读取</h2 >
        ${cookie.userType.name } === ${cookie.userType.value }< br >
        ${cookie["userType"].name } === ${cookie["userType"].value }
    </body >
    </html >
```

输入网址 http://localhost：8080/0503-FieldsDataServlet/CookieVarServlet 运行 CookieVarServlet，首次运行可能会出现 java.lang.NullPointerException 空指针异常，这是因为 cookie 的写入略有延迟，只需重新刷新浏览器，即可得到预期结果，如图 5-6 所示。

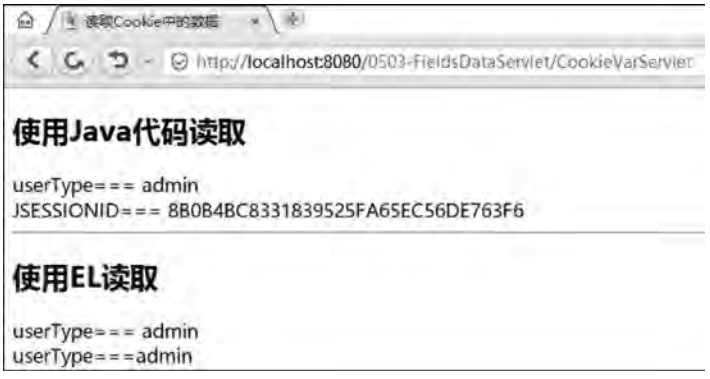

图 5-6　读取 cookie 数据

EL 中代码 ${cookie.userType.name} 表示输出 cookie 中名字为 userType 变量的 name 属性值，其实还是 userType；${cookie.userType.value} 则是输出名为 userType 的变量的值，此处为 admin。使用 cookie 变量还可以访问当前会话 cookie 的 ID 值，例如：

```
${cookie.JSESSIONID.value}
```

5. header 和 headerValues 对象

header 和 headerValues 是用于获取 HTTP 请求消息头的隐式对象。当客户端浏览器发送一个请求时，header 和 headerValues 变量从 HTTP 请求头中检索值，它们的运行机制与 param 和 paramValues 类似。可以使用 ${header.name} 或 ${header["name"]} 从 header 对象中读取名字为 name 的变量值；使用 ${headerValues.name[0]}、${headerValues.name["0"]} 或 ${headerValues.name['0']} 来访问 header 中名为 name 的数组首元素。例如，下列代码返回请求头的 host 值。

```
${header.host} 或 ${header["host"]}
${headerValues.host[0]}、${headerValues.host["0"]} 或 ${headerValues.host['0']}
```

6. initParam 对象

initParam 是用于获取 Web 应用初始化信息的隐式对象，这些参数称作 Web 工程初始

参数。这些初始化参数存储在整个项目 WebContent\WEB-INF 目录下的 web.xml 配置文件中，在 JSP 中是通过 ServletContext 对象的 getInitParameter(String name)函数获取参数值，EL 则是通过 ${initParam.name}表达式获取。

【例 5-6】 使用 EL 表达式获取 Web 项目初始化参数。

(1) 在 Eclipse 中新建一个名为 0506-initParamTest 的项目，注意创建相应的 web.xml 配置文件。

(2) 编辑 web.xml 配置文件，加入测试参数。

(3) 新建 initParamTest.jsp 页面，使用 JSP 代码和 EL 表达式两种方式获取初始化参数的值。

① web.xml 文件内容。

```
<?xml version = "1.0" encoding = "UTF - 8"?>
< web - app xmlns:xsi = "http://www.w3.org/2001/XMLSchema - instance"
    xmlns = "http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation = "http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web - app_3_1.xsd"
    version = "3.1">
    < display - name > 0503 - FieldsDataServlet </display - name >
    < welcome - file - list >
        < welcome - file > index.html </welcome - file >
        < welcome - file > index.htm </welcome - file >
        < welcome - file > index.jsp </welcome - file >
        < welcome - file > default.html </welcome - file >
        < welcome - file > default.htm </welcome - file >
        < welcome - file > default.jsp </welcome - file >
    </welcome - file - list >

    < context - param >
        < param - name > SysCnName </param - name >
        < param - value >南京工业大学资产管理系统</param - value >
    </context - param >
    < context - param >
        < param - name > SysVer </param - name >
        < param - value > 1.2.8 </param - value >
    </context - param >

</web - app >
```

在配置文件中，加入了 SysCnName 和 SysVer 两个初始化参数，值分别为“南京工业大学资产管理系统”和“1.2.8”。

② initParms.jsp 页面源代码。

```
< % @ page language = "java" contentType = "text/html; charset = UTF - 8"
    pageEncoding = "UTF - 8" % >
<! DOCTYPE html PUBLIC " - //W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
< html >
< head >
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>测试 Web 项目初始化参数</title>
</head>
<body>
  <h2>使用 JSP 代码获取</h2>
  <%
      out.println(request.getServletContext().getInitParameter("SysCnName"));
      out.println("<br>");
      out.println(request.getServletContext().getInitParameter("SysVer"));
   %>
  <hr>
  <h2>使用 EL 表达式获取</h2>
   ${initParam.SysCnName }<br>
   ${initParam.SysVer }<br>

</body>
</html>
```

测试结果如图 5-7 所示。

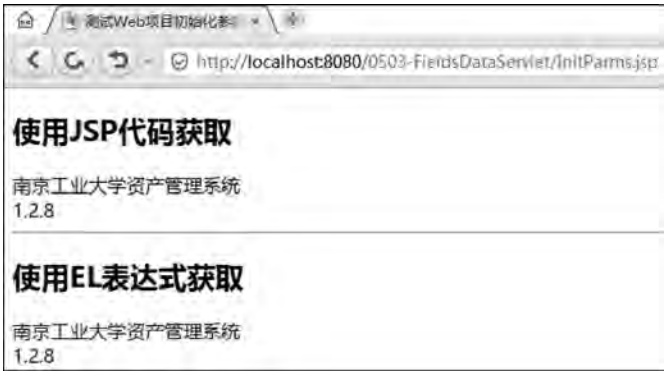

图 5-7　获取 initParam 初始化参数值

5.1.4　EL 的数据访问

使用 EL 除了可以方便地访问隐式对象中的数据之外，还可以轻松读取作用域变量、JavaBeans 的属性以及集合的元素值。其中部分内容在 5.1.3 节已经涉及，此处进行总结和梳理。

1. 访问作用域变量

Web 程序中的作用域按范围由小到大的顺序依次为 page 页面、request 请求、session 会话和 application 应用。访问这些作用域中变量数据的一般做法是：在 Servlet 或 JSP 中使用 Java 代码调用特定域对象的 setAttribute("var",varValue)函数写入数据到对应的作用域；然后使用 RequestDispatcher 对象的 forward()函数将请求转发到另一个 JSP 页面；在转发到的 JSP 页面中使用脚本表达式调用域对象的 getAtrribute("var")函数或 EL 表达式的 ${var}读取变量值，如表 5-10 所示。

表 5-10　访问作用域变量

作用域	写　　入	JSP 读取	EL 读取
page 页面	PageContext. setAttribute("var", varValue)	<%=var%>	${var}
request 请求	HttpServletRequest. setAttribute("var", varValue)		
session 会话	HttpSession. setAttribute("var", varValue)		
application 应用	ServletContext. setAttribute("var", varValue)		

使用表 5-10 中的 ${var}表达式读取域范围的变量值时，Web 容器会依次在页面作用域、请求作用域、会话作用域和应用作用域查找名为 var 的属性。如果找到该属性，则调用其 toString()方法返回对应的属性值，如果找不到则返回空字符串而不是 null，所以使用 EL 表达式读取作用域中的属性值，其返回值必定为字符串型。

在上述 4 个作用域写入变量时，应避免变量同名，否则 EL 读取时先搜索到哪一个变量，就返回相应的值。若确有需要在不同作用域中写入同名变量，那么在读取时则应添加对应的作用域修饰语 ${pageScope. var}、${requestScope. var}、${sessionScope. var}或 ${applicationScope. var}加以区分。

2. 访问 JavaBean 属性

在实际的应用开发中，通常把项目的业务逻辑放在 Servlet 中进行处理，由 Servlet 实例化 JavaBean 对象，再通过指定的 JSP 程序显示 JavaBean 中的内容。

【例 5-7】 使用 EL 表达式访问 JavaBean。

(1) 新建 Java Web 项目 0507-JavaBeanDemo。

(2) 新建两个 JavaBean 类 Address 和 Teacher。

(3) 新建 teacherDemo. jsp 用来显示 JavaBean 的对象内容。

(4) 新建 Servlet 类 TeacherServlet，在 doGet()方法中初始化 Teacher 类对象，并转发到 teacherDemo. jsp 页面。

详细的代码文件如下。

① Address. java 源码。

```
package cn.pju.beans;

public class Address {
  private String city;
  private String zipCode;

  public Address() {
      super();
  }
  public Address(String city, String zipCode) {
      super();
      this.city = city;
      this.zipCode = zipCode;
  }

  public String getCity() {
      return city;
```

```
    }
    public void setCity(String city) {
        this.city = city;
    }
    public String getZipCode() {
        return zipCode;
    }
    public void setZipCode(String zipCode) {
        this.zipCode = zipCode;
    }
}
```

② Teacher.java 源码。

```
package cn.pju.beans;

public class Teacher {
    private String tName;
    private Address address;

    public Teacher() {
        super();
    }
    public Teacher(String tName, Address address) {
        super();
        this.tName = tName;
        this.address = address;
    }

    public String gettName() {
        return tName;
    }
    public void settName(String tName) {
        this.tName = tName;
    }
    public Address getAddress() {
        return address;
    }
    public void setAddress(Address address) {
        this.address = address;
    }
}
```

③ TeacherServlet.java 源码。

```
package cn.pju.servlets;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;

import cn.pju.beans.Address;
import cn.pju.beans.Teacher;

@WebServlet("/TeacherServlet")
public class TeacherServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

    public TeacherServlet() {
        super();
    }

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      response.setCharacterEncoding("UTF-8");
      Address address = new Address("南京市", "210088");
      Teacher lina = new Teacher("李娜", address);
      request.setAttribute("teacher", lina);
      request.getRequestDispatcher("/teacherDemo.jsp").forward(request, response);
  }

  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      doGet(request, response);
  }

}
```

④ teacherDemo.jsp 页面文件源码。

```
<%@page import="cn.pju.beans.Teacher" %>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/
loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>EL 访问 JavaBean</title>
</head>
<body>
  <h2>使用 Java 代码</h2>
  <%  Teacher ln = (Teacher)request.getAttribute("teacher");  %>
  <%=ln %><br>
  <%=ln.gettName() %><br>
  <%=ln.getAddress().getCity() %><br>
  <%=ln.getAddress().getZipCode() %>
  <%ln.settName("林娜"); %><br>
  <%=ln.gettName() %><br>
```

```
<h2>使用 JSP 标准动作</h2>
<jsp:useBean id="teacher" class="cn.pju.beans.Teacher" scope="request"></jsp:useBean>
<jsp:getProperty property="tName" name="teacher"/><br>
<%=teacher.gettName()%><br>
<%=teacher.getAddress().getCity()%><br>
<%=teacher.getAddress().getZipCode()%><br>
<jsp:setProperty property="tName" name="teacher" value="琳娜"/>

<h2>使用 EL</h2>
${teacher}<br>
${teacher.tName}<br>
${teacher.address}<br>
${teacher.address.city}<br>
${teacher.address.zipCode}<br>

</body>
</html>
```

3. 访问集合元素

EL 可以访问 Array 数组、List 序列、Vector 向量和 Map 对象等集合元素中的数据，存储在集合中的元素可以是普通类型变量，也可以是对象型数据。EL 访问集合元素时需要使用集合元素访问运算符“[]”，其语法格式为：${collection[index]}。

(1) 获取数组中的元素。

```
<%
    String[] arr =  {"Java","Python","R"};
    pageContext.setAttribute("arr", arr);
%>
```

获取数组中的元素可以有以下 3 种写法。

```
${arr[0]}
${arr["1"]}
${arr['2']}
```

(2) 输出 List 中指定索引位置的元素。

```
<%
    List list = new ArrayList<String>();
    list.add("红楼梦");
    list.add("西游记");
    list.add("水浒传");
    list.add("三国演义");
    request.setAttribute("list", list);
%>
${list[0]}<br>
${list["1"]}<br>
${list['2']}<br>
${list[3]}<br>
```

(3) 输出 Map 中指定键对应的值。

由于 Map 是典型的键值对(key-value)序列,读取对应数据时需要指定要访问的 key 值,而不是对应的索引值 index,所以有 ${collection.key} 和 ${collection["key"]} 两种写法。

```
<%
  Map stu = new HashMap();
  stu.put("sno","00002");
  stu.put("sname","郑炫君");
  session.setAttribute("stu",stu);
%>
${stu.sno}<br>
${stu["sname"]}<br>
```

(4) 读取 Vector 向量中的元素。

集合元素中不仅可以存放普通型变量,也可以存放对象型数据,为此我们定义一个学生类 Student,代码如下。

```
package cn.pju.pojo;

public class Student {
  private String sno;
  private String sname;

  public Student() {
      super();
  }
  public Student(String sno, String sname) {
      super();
      this.sno = sno;
      this.sname = sname;
  }

  public String getSno() {
      return sno;
  }
  public void setSno(String sno) {
      this.sno = sno;
  }
  public String getSname() {
      return sname;
  }
  public void setSname(String sname) {
      this.sname = sname;
  }
}
```

下面是在 JSP 中读取向量中数据的代码。

```
<%
  Vector<Student> vtr = new Vector<Student>();
  Student ll = new Student("001", "李雷");
  Student ksh = new Student("002", "柯世怀");
  vtr.addElement(ll);
  vtr.addElement(ksh);
  application.setAttribute("vtr",vtr);
%>
${vtr[0].sno}<br>
${vtr[0].sname}<br>
${vtr["1"].sno}<br>
${vtr['1'].sname}<br>
```

4. 访问 EL 的隐含对象

EL 访问对应 11 个隐含对象中属性的知识在 5.1.3 节中已经做了详细讲解，此处不再重复。

视频讲解

5.2 JSTL 标准标签库

JSTL 的全称是 Java Server Pages Standard Tag Library(JSP 标准标签库)，是一个不断完善的开放源代码的 JSP 标准标签库，由 Apache 的 Jakarta 小组负责维护，主要是给 Java Web 开发人员提供一个标准的通用标签库。JSTL 可以应用于各种领域，如基本输入输出、流程控制、循环、XML 文件解析、数据库查询及国际化和文字格式标准化等应用。JSTL 的引入可以取代传统 JSP 程序中嵌入 Java 代码的做法，大大提高了程序的可维护性。

1. JSTL 的逻辑组成

JSTL 包含四个标签库和一组 EL 函数。为方便用户使用，JSP 规范中描述了 JSTL 的各个标签库的 URI 地址和建议使用的前缀名，如表 5-11 所示。本章中在使用 JSTL 标签时，使用的都是这些建议的前缀。要在 JSP 页面中使用 JSTL，必须添加 taglib 指令，其作用是引入需要使用的标签库，并在该 JSP 文件中进行声明(与变量的声明和引用类似，只有声明后的标签库才可以使用)，同时指定标签的前缀(类似于别名，可以简化编程脚本)。表 5-11 中的 URI(Universal Resource Identifier，统一资源标识符)表示标签的网络资源位置，前缀由 prefix 关键字引出，表中给出的均为默认前缀，类似于简写别名，用户可以自主修改。

表 5-11 JSTL 标签函数库

标签库	前缀	JRL	库 功 能	示 例
core 核心标签库	c	http://java.sun.com/jsp/jstl/core	操作范围变量、流程控制、URL 操作	<c:out>
I18N 国际化标签库	fmt	http://java.sun.com/jsp/jstl/fmt	操作通过 XML 表示的数据	<fmt:formatDate>
SQL 标签库	sql	http://java.sun.com/jsp/jstl/sql	数字及日期数据格式化、页面国际化	<sql:query>
XML 标签库	x	http://java.sun.com/jsp/jstl/xml	操作关系数据库	<x:forEach>
函数标签库	fn	http://java.sun.com/jsp/jstl/functions	字符串处理	<fn:split>

以声明 core 核心标签库为例，其基本语法如下。

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

上述代码表示在当前 JSP 页面中引入 JSTL 的核心标签库，并且以“c”作为访问用前缀别名。需要说明的是，在 JSP 页面中引入 taglib 指令，一般位于 page 指令之后。

2. JSTL 的物理组成

完整的 JSTL 应包含 Sun 公司提供的 jstl.jar 包和 Web 容器生产商提供的 JSTL 实现包，以 Apache Jakarta 小组提供的 JSTL 实现包为例，完整的 JSTL 包含 jstl.jar、standard.jar 和 xalan.jar 三个 jar 包。Sun 公司提供的 jstl.jar 包封装了 JSTL 所要求的一些 API 和类，Apache Jakarta 小组编写的 JSTL API 实现类封装在 standard.jar 包中。standard.jar 包中包括核心标签库、国际化/格式化标签库、数据库标签库中的标签和标准的 EL 自定义函数的实现类，xalan.jar 包中包括 JSTL 解析 XPath 的相关 API 类。

3. JSTL 的使用

在 Eclipse 下创建 JSP 页面，使用 JSTL 标签的详细步骤如下。

(1) 上网下载 JSTL 的 jar 包，需要同时下载 jstl.jar 和 standard.jar，具体下载地址如下。

jstl-1.2 版本下载地址为：http://central.maven.org/maven2/jstl/jstl/1.2/jstl-1.2.jar。

standard-1.1.2 版本下载地址为：http://central.maven.org/maven2/taglibs/standard/1.1.2/standard-1.1.2.jar。

以上资源笔者都是下载于 Maven 仓库官方网址 https://mvnrepository.com/tags/maven，读者可进入该网址免费搜索下载所需版本的相关 jar 包，也可以进入 Apache 官网搜索下载。

(2) 在 Eclipse 中新建一个 Dynamic Web Project，将下载好的 jstl-1.2.jar 和 standard-1.1.2.jar 两个文件复制到项目中 WebContent/WEB-INF/lib 目录下。

(3) 新建 JSP 页面，在页面的 page 指令后添加 taglib 指令，引入 JSTL 库，prefix 和 uri 的属性值参见表 5-11。例如，引入核心标签库的 taglib 指令为：

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

(4) 在 JSP 页面中直接使用标签，例如：

```
<c:out value="${5 + 3}"></c:out>
```

扩展阅读

小　结

表达式语言(EL)是 JSP 2.0 之后增加的新特征,其目标是使动态网页的设计、开发和维护更加简单容易。EL 只是一种数据访问语言,通过它可以很方便地在 JSP 页面中访问应用程序数据,无须使用小脚本和请求时表达式,程序员甚至可以不用学习 Java 语言就可以使用表达式语言。

表达式语言最重要的目的是创建无脚本的 JSP 页面,为了实现这个目的,EL 定义了自己的运算符和语法等,它完全能够取代传统 JSP 中的声明、表达式和小脚本。

习　题

1. 简述表达式语言的主要功能。
2. 属性与集合访问运算符中的点(.)运算符和方括号([])运算符有什么区别?
3. 在 EL 中,都可以访问哪些类型的数据?