

# 09-EL 表达式 & JSTL 标签库

讲师：王振国

## 今日任务

### 1.EL 表达式

#### a)什么是 EL 表达式，EL 表达式的作用？

EL 表达式的全称是：Expression Language。是表达式语言。

EL 表达式的什么作用：EL 表达式主要是代替 jsp 页面中的表达式脚本在 jsp 页面中进行数据的输出。  
因为 EL 表达式在输出数据的时候，要比 jsp 的表达式脚本要简洁很多。

```
<body>
<%
    request.setAttribute("key","值");
%>
表达式脚本输出 key 的值是：
<%=request.getAttribute("key1")==null?"":request.getAttribute("key1")%><br/>
EL 表达式输出 key 的值是：${key1}
</body>
```

EL 表达式的格式是：\${表达式}

EL 表达式在输出 null 值的时候，输出的是空串。jsp 表达式脚本输出 null 值的时候，输出的是 null 字符串。

#### b)EL 表达式搜索域数据的顺序

EL 表达式主要是在 jsp 页面中输出数据。

主要是输出域对象中的数据。

当四个域中都有相同的 key 的数据的时候，EL 表达式会按照四个域的从小到大的顺序去进行搜索，找到就输出。

```
<body>
<%
    //往四个域中都保存了相同的key的数据。
    request.setAttribute("key", "request");
%>
```

```
session.setAttribute("key", "session");
application.setAttribute("key", "application");
pageContext.setAttribute("key", "pageContext");
%>
${ key }
```

</body>

## c)EL 表达式输出 Bean 的普通属性，数组属性。List 集合属性，map 集合属性

- i. 需求——输出 Person 类中普通属性，数组属性。list 集合属性和 map 集合属性。

Person 类

```
public class Person {
//    i.需求——输出 Person 类中普通属性，数组属性。list 集合属性和map 集合属性。
    private String name;
    private String[] phones;
    private List<String> cities;
    private Map<String,Object> map;

    public int getAge() {
        return 18;
    }
}
```

输出的代码:

```
<body>
<%
    Person person = new Person();
    person.setName("国哥好帅! ");
    person.setPhones(new String[]{"18610541354", "18688886666", "18699998888"});

    List<String> cities = new ArrayList<String>();
    cities.add("北京");
    cities.add("上海");
    cities.add("深圳");
    person.setCities(cities);

    Map<String,Object>map = new HashMap<>();
    map.put("key1", "value1");
    map.put("key2", "value2");
    map.put("key3", "value3");
    person.setMap(map);
}
```

```
pageContext.setAttribute("p", person);
```

```
%>
```

输出 Person: `${ p }`<br/>

输出 Person 的 name 属性: `${p.name}` <br>

输出 Person 的 phones 数组属性值: `${p.phones[2]}` <br>

输出 Person 的 cities 集合中的元素值: `${p.cities}` <br>

输出 Person 的 List 集合中个别元素值: `${p.cities[2]}` <br>

输出 Person 的 Map 集合: `${p.map}` <br>

输出 Person 的 Map 集合中某个 key 的值: `${p.map.key3}` <br>

输出 Person 的 age 属性: `${p.age}` <br>

```
</body>
```

## d)EL 表达式——运算

语法: `${ 运算表达式 }` , EL 表达式支持如下运算符:

### 1) 关系运算

关系运算符	说 明	范 例	结果
<code>==</code> 或 <code>eq</code>	等于	<code>\${ 5 == 5 }</code> 或 <code>\${ 5 eq 5 }</code>	true
<code>!=</code> 或 <code>ne</code>	不等于	<code>\${ 5 != 5 }</code> 或 <code>\${ 5 ne 5 }</code>	false
<code>&lt;</code> 或 <code>lt</code>	小于	<code>\${ 3 &lt; 5 }</code> 或 <code>\${ 3 lt 5 }</code>	true
<code>&gt;</code> 或 <code>gt</code>	大于	<code>\${ 2 &gt; 10 }</code> 或 <code>\${ 2 gt 10 }</code>	false
<code>&lt;=</code> 或 <code>le</code>	小于等于	<code>\${ 5 &lt;= 12 }</code> 或 <code>\${ 5 le 12 }</code>	true
<code>&gt;=</code> 或 <code>ge</code>	大于等于	<code>\${ 3 &gt;= 5 }</code> 或 <code>\${ 3 ge 5 }</code>	false

### 2) 逻辑运算

逻辑运算符	说 明	范 例	结果
<code>&amp;&amp;</code> 或 <code>and</code>	与运算	<code>\${ 12 == 12 &amp;&amp; 12 &lt; 11 }</code> 或 <code>\${ 12 == 12 and 12 &lt; 11 }</code>	false
<code>  </code> 或 <code>or</code>	或运算	<code>\${ 12 == 12    12 &lt; 11 }</code> 或 <code>\${ 12 == 12 or 12 &lt; 11 }</code>	true
<code>!</code> 或 <code>not</code>	取反运算	<code>\${ !true }</code> 或 <code>\${ not true }</code>	false

### 3) 算数运算

算数运算符	说 明	范 例	结果
-------	-----	-----	----

+	加法	<code>\${ 12 + 18 }</code>	30
-	减法	<code>\${ 18 - 8 }</code>	10
*	乘法	<code>\${ 12 * 12 }</code>	144
/ 或 div	除法	<code>\${ 144 / 12 }</code> 或 <code>\${ 144 div 12 }</code>	12
% 或 mod	取模	<code>\${ 144 % 10 }</code> 或 <code>\${ 144 mod 10 }</code>	4

## i. empty 运算

empty 运算可以判断一个数据是否为空，如果为空，则输出 true,不为空输出 false。

以下几种情况为空：

- 1、值为 null 值的时候，为空
- 2、值为空串的时候，为空
- 3、值是 Object 类型数组，长度为零的时候
- 4、list 集合，元素个数为零
- 5、map 集合，元素个数为零

```
<body>
  <%
//      1、值为null 值的时候，为空
    request.setAttribute("emptyNull", null);
//      2、值为空串的时候，为空
    request.setAttribute("emptyStr", "");
//      3、值是Object 类型数组，长度为零的时候
    request.setAttribute("emptyArr", new Object[]{});
//      4、List 集合，元素个数为零
    List<String> list = new ArrayList<>();
//      list.add("abc");
    request.setAttribute("emptyList", list);
//      5、map 集合，元素个数为零
    Map<String, Object> map = new HashMap<String, Object>();
//      map.put("key1", "value1");
    request.setAttribute("emptyMap", map);
  %>
  ${ empty emptyNull } <br/>
  ${ empty emptyStr } <br/>
  ${ empty emptyArr } <br/>
  ${ empty emptyList } <br/>
  ${ empty emptyMap } <br/>
</body>
```

## ii. 三元运算

表达式 1? 表达式 2: 表达式 3

如果表达式 1 的值为真，返回表达式 2 的值，如果表达式 1 的值为假，返回表达式 3 的值。

示例:

```
${ 12 != 12 ? "国哥帅呆":"国哥又骗人啦" }
```

### iii. "."点运算 和 [] 中括号运算符

.点运算，可以输出 Bean 对象中某个属性的值。

[]中括号运算，可以输出有序集合中某个元素的值。

并且[]中括号运算，还可以输出 map 集合中 key 里含有特殊字符的 key 的值。

```
<body>
  <%
    Map<String,Object> map = new HashMap<String, Object>();
    map.put("a.a.a", "aaaValue");
    map.put("b+b+b", "bbbValue");
    map.put("c-c-c", "cccValue");

    request.setAttribute("map", map);
  %>

  ${ map['a.a.a'] } <br>
  ${ map["b+b+b"] } <br>
  ${ map['c-c-c'] } <br>
</body>
```

## e)EL 表达式的 11 个隐含对象

EL 个达式中 11 个隐含对象，是 EL 表达式中自己定义的，可以直接使用。

变量	类型	作用
pageContext	PageContextImpl	它可以获取 jsp 中的九大内置对象
pageScope	Map<String,Object>	它可以获取 pageContext 域中的数据
requestScope	Map<String,Object>	它可以获取 Request 域中的数据
sessionScope	Map<String,Object>	它可以获取 Session 域中的数据
applicationScope	Map<String,Object>	它可以获取 ServletContext 域中的数据
param	Map<String,String>	它可以获取请求参数的值
paramValues	Map<String,String[]>	它也可以获取请求参数的值，获取多个值的时候使用。
header	Map<String,String>	它可以获取请求头的信息
headerValues	Map<String,String[]>	它可以获取请求头的信息，它可以获取多个值的情况
cookie	Map<String,Cookie>	它可以获取当前请求的 Cookie 信息

## i. EL 获取四个特定域中的属性

pageScope	=====	pageContext 域
requestScope	=====	Request 域
sessionScope	=====	Session 域
applicationScope	=====	ServletContext 域

```
<body>
  <%
    pageContext.setAttribute("key1", "pageContext1");
    pageContext.setAttribute("key2", "pageContext2");
    request.setAttribute("key2", "request");
    session.setAttribute("key2", "session");
    application.setAttribute("key2", "application");
  %>
  ${ applicationScope.key2 }
</body>
```

## ii. pageContext 对象的使用

1. 协议:
2. 服务器 ip:
3. 服务器端口:
4. 获取工程路径:
5. 获取请求方法:
6. 获取客户端 ip 地址:
7. 获取会话的 id 编号:

```
<body>
  <!--
    request.getScheme() 它可以获取请求的协议
    request.getServerName() 获取请求的服务器 ip 或域名
    request.getServerPort() 获取请求的服务器端口号
    getContextPath() 获取当前工程路径
    request.getMethod() 获取请求的方式 (GET 或 POST)
    request.getRemoteHost() 获取客户端的 ip 地址
    session.getId() 获取会话的唯一标识
  -->
  <%
    pageContext.setAttribute("req", request);
  %>
  <%=request.getScheme() %> <br>
```

1. 协议: `${ req.scheme }`<br>
2. 服务器 ip: `${ pageContext.request.serverName }`<br>
3. 服务器端口: `${ pageContext.request.serverPort }`<br>
4. 获取工程路径: `${ pageContext.request.contextPath }`<br>
5. 获取请求方法: `${ pageContext.request.method }`<br>
6. 获取客户端 ip 地址: `${ pageContext.request.remoteHost }`<br>
7. 获取会话的 id 编号: `${ pageContext.session.id }`<br>

&lt;/body&gt;

### iii. EL 表达式其他隐含对象的使用

param	Map<String,String>
paramValues	Map<String,String[]>

它可以获取请求参数的值

它也可以获取请求参数的值，获取多个值的时候使用。

示例代码:

```
输出请求参数 username 的值: ${ param.username } <br>
输出请求参数 password 的值: ${ param.password } <br>
```

```
输出请求参数 username 的值: ${ paramValues.username[0] } <br>
输出请求参数 hobby 的值: ${ paramValues.hobby[0] } <br>
输出请求参数 hobby 的值: ${ paramValues.hobby[1] } <br>
```

请求地址:

```
http://localhost:8080/09_EL_JSTL/other_el_obj.jsp?username=wzg168&password=666666&hobby=java&hobby=cpp
```

header	Map<String,String>
headerValues	Map<String,String[]>

它可以获取请求头的信息

它可以获取请求头的信息，它可以获取多个值的情况

示例代码:

```
输出请求头【User-Agent】的值: ${ header['User-Agent'] } <br>
输出请求头【Connection】的值: ${ header.Connection } <br>
输出请求头【User-Agent】的值: ${ headerValues['User-Agent'][0] } <br>
```

cookie	Map<String,Cookie>
--------	--------------------

它可以获取当前请求的 Cookie 信息

示例代码:

```
获取 Cookie 的名称: ${ cookie.JSESSIONID.name } <br>
获取 Cookie 的值: ${ cookie.JSESSIONID.value } <br>
```

initParam

Map&lt;String,String&gt;

它可以获取在 web.xml 中配置的&lt;context-param&gt;上下文参数

web.xml 中的配置:

```
<context-param>
  <param-name>username</param-name>
  <param-value>root</param-value>
</context-param>

<context-param>
  <param-name>url</param-name>
  <param-value>jdbc:mysql:///test</param-value>
</context-param>
```

示例代码:

```
输出<code>&lt;Context-param&gt;username 的值: ${ initParam.username } <br>
输出<code>&lt;Context-param&gt;url 的值: ${ initParam.url } <br>
```

## 2、JSTL 标签库(次重点\*\*\*\*)

JSTL 标签库 全称是指 JSP Standard Tag Library JSP 标准标签库。是一个不断完善的开放源代码的 JSP 标签库。

EL 表达式主要是为了替换 jsp 中的表达式脚本，而标签库则是为了替换代码脚本。这样使得整个 jsp 页面变得更佳简洁。

JSTL 由五个不同功能的标签库组成。

功能范围	URI	前缀
核心标签库--重点	<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>	c
格式化	<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>	fmt
函数	<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>	fn
数据库(不使用)	<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>	sql
XML(不使用)	<a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>	x

在 jsp 标签库中使用 taglib 指令引入标签库

CORE 标签库

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

XML 标签库



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

FMT 标签库

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

SQL 标签库

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

FUNCTIONS 标签库

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

## f) JSTL 标签库的使用步骤

- 1、先导入 jstl 标签库的 jar 包。

taglibs-standard-impl-1.2.1.jar

taglibs-standard-spec-1.2.1.jar

- 2、第二步，使用 taglib 指令引入标签库。

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

## g)core 核心库使用

### i. <c:set /> （使用很少）

作用：set 标签可以往域中保存数据

```
<%--
```

```
i.<c:set />
```

作用：set 标签可以往域中保存数据

域对象.setAttribute(key,value);

scope 属性设置保存到哪个域

page 表示PageContext 域（默认值）

request 表示Request 域

session 表示Session 域

application 表示ServletContext 域

var 属性设置 key 是多少

value 属性设置值

```
--%>
```

保存之前: `${ sessionScope.abc }` <br>

```
<c:set scope="session" var="abc" value="abcValue"/>
```

保存之后: `${ sessionScope.abc }` <br>

## ii. <c:if />

if 标签用来做 if 判断。

```
<%--
    ii.<c:if />
    if 标签用来做 if 判断。
    test 属性表示判断的条件（使用 EL 表达式输出）
--%>
<c:if test="${ 12 == 12 }">
    <h1>12 等于 12</h1>
</c:if>
<c:if test="${ 12 != 12 }">
    <h1>12 不等于 12</h1>
</c:if>
```

## iii. <c:choose> <c:when> <c:otherwise> 标签

作用：多路判断。跟 switch ... case .... default 非常接近

```
<%--
iii.<c:choose> <c:when> <c:otherwise> 标签
作用：多路判断。跟 switch ... case .... default 非常接近

choose 标签开始选择判断
when 标签表示每一种判断情况
    test 属性表示当前这种判断情况的值
otherwise 标签表示剩下的情况

<c:choose> <c:when> <c:otherwise> 标签使用时需要注意的点：
    1、标签里不能使用 html 注释，要使用 jsp 注释
    2、when 标签的父标签一定要是 choose 标签
--%>
<%
    request.setAttribute("height", 180);
%>
<c:choose>
    <%-- 这是 html 注释 --%>
    <c:when test="${ requestScope.height > 190 }">
        <h2>小巨人</h2>
    </c:when>
    <c:when test="${ requestScope.height > 180 }">
        <h2>很高</h2>
    </c:when>
    <c:when test="${ requestScope.height > 170 }">
        <h2>还可以</h2>
    </c:when>
```

```
<c:otherwise>
  <c:choose>
    <c:when test="${requestScope.height > 160}">
      <h3>大于 160</h3>
    </c:when>
    <c:when test="${requestScope.height > 150}">
      <h3>大于 150</h3>
    </c:when>
    <c:when test="${requestScope.height > 140}">
      <h3>大于 140</h3>
    </c:when>
    <c:otherwise>
      其他小于 140
    </c:otherwise>
  </c:choose>
</c:otherwise>
</c:choose>
```

#### iv. <c:forEach />

作用：遍历输出使用。

##### 1. 遍历 1 到 10，输出

示例代码：

```
<!-- 1. 遍历 1 到 10，输出
begin 属性设置开始的索引
end 属性设置结束的索引
var 属性表示循环的变量(也是当前正在遍历到的数据)
for (int i = 1; i < 10; i++)
--%>
<table border="1">
  <c:forEach begin="1" end="10" var="i">
    <tr>
      <td>第${i}行</td>
    </tr>
  </c:forEach>
</table>
```

##### 2. 遍历 Object 数组

示例代码：

```
<!-- 2. 遍历 Object 数组
    for (Object item: arr)
        items 表示遍历的数据源（遍历的集合）
        var 表示当前遍历到的数据
--%>
<%
    request.setAttribute("arr", new String[]{"18610541354", "18688886666", "18699998888"});
%>
<c:forEach items="${ requestScope.arr }" var="item">
    ${ item } <br>
</c:forEach>
```

### 3. 遍历 Map 集合

示例代码:

```
<%
    Map<String,Object> map = new HashMap<String, Object>();
    map.put("key1", "value1");
    map.put("key2", "value2");
    map.put("key3", "value3");
//    for ( Map.Entry<String,Object> entry : map.entrySet()) {
//    }
    request.setAttribute("map", map);
%>
<c:forEach items="${ requestScope.map }" var="entry">
    <h1>${entry.key} = ${entry.value}</h1>
</c:forEach>
```

### 4. 遍历 List 集合---list 中存放 Student 类，有属性：编号，用户名，密码，年龄，电话信息

Student 类:

```
public class Student {
    //4. 编号，用户名，密码，年龄，电话信息
    private Integer id;
    private String username;
    private String password;
    private Integer age;
    private String phone;
```

示例代码:

```
<!--4.遍历List 集合---List 中存放 Student 类，有属性：编号，用户名，密码，年龄，电话信息-->
<%
    List<Student> studentList = new ArrayList<Student>();
    for (int i = 1; i <= 10; i++) {
        studentList.add(new Student(i,"username"+i ,"pass"+i,18+i,"phone"+i));
    }
    request.setAttribute("stus", studentList);
%>
<table>
    <tr>
        <th>编号</th>
        <th>用户名</th>
        <th>密码</th>
        <th>年龄</th>
        <th>电话</th>
        <th>操作</th>
    </tr>
    <!--
        items 表示遍历的集合
        var 表示遍历到的数据
        begin 表示遍历的开始索引值
        end 表示结束的索引值
        step 属性表示遍历的步长值
        varStatus 属性表示当前遍历到的数据的状态
        for (int i = 1; i < 10; i+=2)
    --%>
<c:forEach begin="2" end="7" step="2" varStatus="status" items="${requestScope.stus}" var="stu">
    <tr>
        <td>${stu.id}</td>
        <td>${stu.username}</td>
        <td>${stu.password}</td>
        <td>${stu.age}</td>
        <td>${stu.phone}</td>
        <td>${status.step}</td>
    </tr>
</c:forEach>
</table>
```