EL表达式&JSTL标签库

EL表达式

介绍和作用

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

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。

```jsp
<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>
```

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

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

```java
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;
    }

包含相对应的getXXX方法、setXXX方法、toString方法、无参和有参构造器
```

```jsp
<body>
    <%
    Person person = new Person();
    person.setName("JunXing");
    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 pnones 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>
    <!--在EL表达式中值的输出是找相对应的get方法-->
    输出 Person 的 age 属性: ${p.age} <br>
</body>
```

EL表达式——运算

关系运算

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	`${ 2 > 10 }` 或 `${ 2 gt 10 }`	false
<= 或 le	小于等于	`${ 5 <= 12 }` 或 `${ 5 le 12 }`	true
>= 或 ge	大于等于	`${ 3 >= 5 }` 或 `${ 3 ge 5}`	false

逻辑运算

逻辑运算符	说 明	范 例	结果

逻辑运算符	说 明	范 例	结果
&&或and	与运算	`${ 12 == 12 && 12 < 11 }` 或 `${ 12 == 12 and 12 < 11 }`	false
||或or	或运算	`${ 12 == 12 \|\| 12 < 11}` 或 `${ 12 == 12 or 12 < 11}`	true
!或not	取反运算	`${ !true}` 或 `${ not true }`	false

算数运算

算数运算符	说 明	范 例	结果
+	加法	${ 12 + 18 }	30
-	减法	${ 18 - 8 }	10
*	乘法	${ 12 * 12 }	144
/或div	除法	`${ 144 / 12 }` 或 `${ 144 div 12 }`	12
%或mod	取模	`${ 144 % 10 }` 或 `${ 144 mod 10 }`	4

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>
```

三元运算

`${表达式 1? 表达式 2: 表达式}`

"."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。

```jsp
<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>
```

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
initParam	Map<String,String>	它可以获取在 web.xml 中配置的上下参数

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

pageContext对象的使用

```
<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>
</body>
```

EL表达式其他隐含对象的使用

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

```
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>
```

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

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

```
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

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

initParam	Map<String,String>	它可以获取在 web.xml 中配置的上下文参数

```
<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>
```

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

JSTL标签库

介绍

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	URL	前缀
核心标签库	http://java.sun.com/jsp/jstl/core	c
格式化	http://java.sun.com/jsp/jstl/fmt	fmt
函数	http://java.sun.com/jsp/jstl/functions	fn
~~数据库~~	http://java.sun.com/jsp/jstl/sql	sql
~~XML~~	http://java.sun.com/jsp/jstl/xml	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" %>
```

JSTL标签库的使用步骤

1. 先导入 jstl 标签库的 jar 包
    a. taglibs-standard-impl-1.2.1.jar
    b. taglibs-standard-spec-1.2.1.jar
2. 使用 taglib 指令引入标签库
    a. `<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`

core核心库的使用

`<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>
```

`<c:if/>`

作用：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）
    if 标签没有 if-else 使用方法
--%>
<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>
```

`<c:choose> <c:when> <c:otherwise>`

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("grade", 80);
%>
<c:choose>
    <c:when test="${ requestScope.grade > 90 }">
        <h2> > 90 </h2>
    </c:when>
    <c:when test="${ requestScope.grade > 80 }">
        <h2> > 80 </h2>
    </c:when>
    <c:when test="${ requestScope.grade > 70 }">
        <h2> > 70 </h2>
    </c:when>
    <c:otherwise>
        <c:choose>
            <c:when test="${requestScope.grade > 60}">
                <h3> > 60 </h3>
            </c:when>
            <c:when test="${requestScope.grade > 50}">
                <h3> > 50 </h3>
            </c:when>
            <c:when test="${requestScope.grade > 40}">
                <h3> > 40 </h3>
            </c:when>
```

```
            <c:otherwise>
                <h3> < 40 </h3>
            </c:otherwise>
        </c:choose>
    </c:otherwise>
</c:choose>
```

`<c:forEach>`

作用：遍历输出使用

```
<%--
    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>
```

使用forEach遍历Object数组

```
<%-- 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[]
{"123","456","789"});
%>
<c:forEach items="${ requestScope.arr }" var="item">
    ${ item } <br>
</c:forEach>
```

使用forEach遍历Map集合

```
<%
    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);
%>
    <%--var只是变量名--%>
<c:forEach items="${ requestScope.map }" var="entry">
    <h1>${entry.key} = ${entry.value}</h1>
</c:forEach>
```

使用forEach遍历LIst集合

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

```
 8      private String phone;
 9
10      public Student() {
11      }
12
13      public Student(Integer id, String username, String password,
    Integer age, String phone) {
14          this.id = id;
15          this.username = username;
16          this.password = password;
17          this.age = age;
18          this.phone = phone;
19      }
20
21      public Integer getId() {
22          return id;
23      }
24
25      public void setId(Integer id) {
26          this.id = id;
27      }
28
29      public String getUsername() {
30          return username;
31      }
32
33      public void setUsername(String username) {
34          this.username = username;
35      }
36
37      public String getPassword() {
38          return password;
39      }
40
41      public void setPassword(String password) {
42          this.password = password;
43      }
44
45      public Integer getAge() {
```

```
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", username='" + username + '\'' +
                ", password='" + password + '\'' +
                ", age=" + age +
                ", phone='" + phone + '\'' +
                '}';
    }
}
```

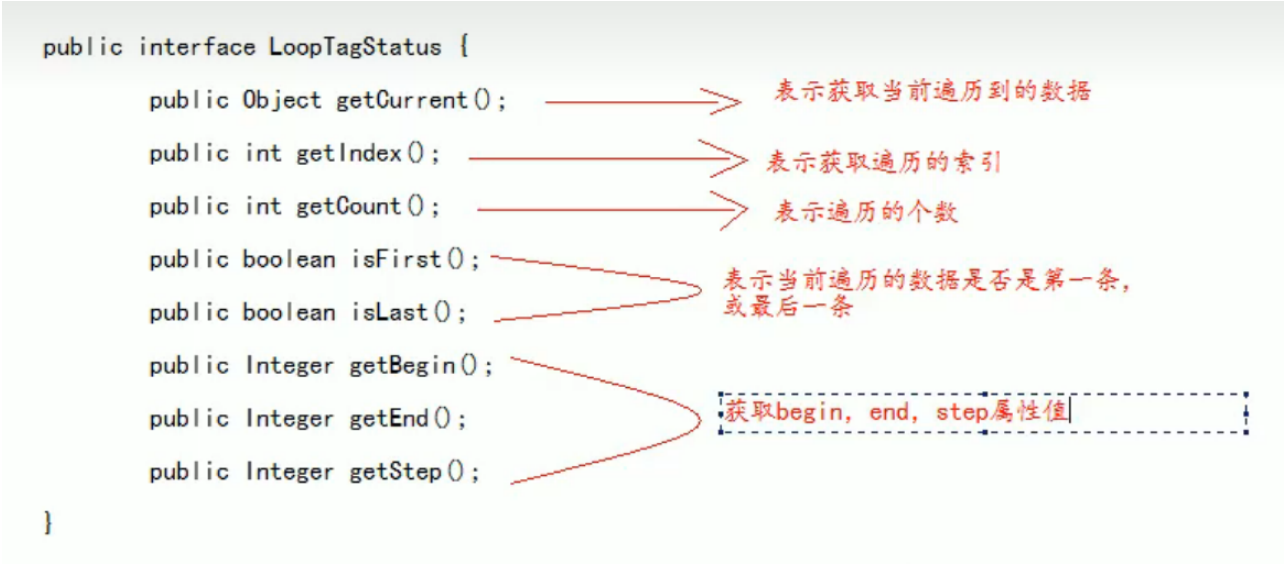
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page import="java.util.ArrayList" %>
<%@ page import="java.util.List" %>
<%@ page import="com.Student" %>
<%--
  Created by IntelliJ IDEA.
  User: JunXing
  Date: 2023/3/22
  Time: 14:52
  To change this template use File | Settings | File Templates.
--%>
```

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    <%
      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
    --%>
    <%-- begin="2" end="7" step="2" varStatus="status" --%>
    <c:forEach 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>

</body>
</html>
```

forEach标签所有所有组合使用介绍




```
<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>
```

返回文首