JSTL

JSTL标签库概述

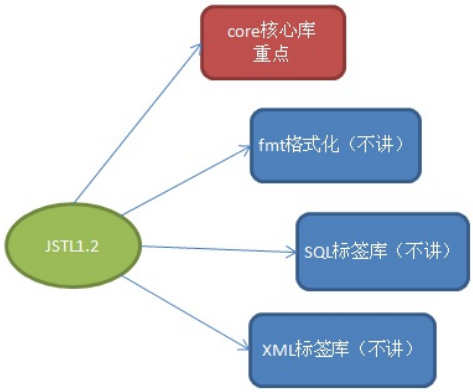
JSTL是apache对EL表达式的扩展（也就是说JSTL依赖EL），JSTL是标签语言！

- 作用：和EL表达式一起 取代<% %>
- 版本：1.0 1.1和1.2（区别不大）
  - 1.0EL表达式没有纳入规范
  - 1.1和1.2EL表达式纳入规范

使用方法

1. 下载jar包，导入到工程中
   - jstl.jar
   - standard.jar

   使用MyEclipse 可以用集成的 jstl



2. 标签库

导入标签库 c.tld 核心的标签库

- JSTL的快速入门
  - 导入相应jar包。
  - 新建JSP的文件，引入标签库 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
  - 直接使用标签库

JSTL的标签

1. <c:out> 输出内容 value ： 输出的内容（常量或者变量） default :默认值 escapeXml：默认是true，进行转义，设置成false，不转义。

```
<c:out value="Hello"></c:out>
<c:out value="${name }"></c:out>
<!-- "" -->
<c:out value="${ city }" default="北京"></c:out>
<c:out value="<a href='#'>超链接</a>" escapeXml="false"/>
        如果不设置escapeXml ， 即为true
        输出到页面中原封不动,不对其进行转移成对应的html元素 <a href="#">超链接</a>
```

2. <c:set>
   - 属性 var ： 定义属性 value ： 存入的值 scope ： 域范围 JavaBean 有关 target ： 修改JavaBean对象 property：修改的属性
   - 向4个web域对象存入值 <c:set var="i" value="10" scope="request" ></c:set>

- 修改JavaBean对象属性的值

```
<% User user=new User(); request.setAttribute("user",user);%>
<c:set target="${user}" property="username"  value="小凤"></c:set>
```

3. <c:remove>
   - 属性 var ： 删除的属性 scope ： 在域的范围
   - 代码 <c:set var="name" value="小凤" scope="page"></c:set> ${ name }

     <c:remove var="name" scope="page"/> ${name }
4. <c:catch>
   - 属性：var 把异常的信息保存变量中
   - 代码 <c:catch var="e"> <% int a = 10/0; %> </c:catch> ${ e.message }
5. <c:if> 没有<c:else>标签
   - 属性
     - test 判断的条件
     - var 计算的结果保存到变量中
     - scope 域的范围
   - 代码 <c:set var="i" value="10" scope="page"></c:set> <c:if test="${ i ge 10 }" var="x" scope="page"> i >= 10 </c:if> <c:if test="${ i lt 10 }"> i < 10 </c:if> ${ x }
6. <c:choose>标签

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

代码：

```
<c:set var="i" value="10" scope="page"></c:set>
<c:choose>
    <c:when test="${ i ge 10 }">
        i >= 10
    </c:when>
    <c:when test="${ i lt 10 }">
        i < 10
    </c:when>
    <c:otherwise>
        其他
    </c:otherwise>
</c:choose>
```

7. <c:forEach>（*****）
   - 循环遍历数据（数组，集合，Map集合）
   - 属性

     var ： 遍历数据的类型 items ： 要遍历的内容

     begin ： 从哪开始 end ： 到哪结束 step ： 步长

     varStatus：记录循环遍历的信息 * index 索引，从0开始 * count（常用） 第多少个，从1计数 * first 是否是集合的第一个元素 * last 是否是几个的最后一个元素 注意：如果是遍历集合、数组、map集合(items标签存在且有值) 则var 对应集合中每一项数据的数据类型 例如 items="map" 则 var="tmep" temp 存储的是 Entry 对象 (temp.key,temp.value)

```
    如果单纯的顺序循环
    例如 :<c:forEach var="i" begin="1" end="100" step="2"></c:forEach>
        则 var 中的变量i  就表示 1, 3(1+2) , 5(3+2)
```

```
  <c:forEach varStatus = "status"/>
      ${status.index}         0
      ${status.count}         1
      ${status.first}         false
```

示例：

```
<%
    String [] arrs = {"美美","小凤","芙蓉","小苍"};
    request.setAttribute("arrs", arrs);
%>
<!-- for(String s : arrs){ }  -->
<c:forEach var="s" items="${ arrs }">
    ${ s }
</c:forEach>

<h4>遍历集合</h4>
<%
    List<String> list = new ArrayList<String>();
    list.add("美美");
    list.add("小凤");
    list.add("芙蓉");
    list.add("小泽");
    request.setAttribute("list", list);
%>
<c:forEach var="s" items="${ list }">
    ${ s }
</c:forEach>

<h4>遍历Map集合</h4>
<%
    Map<String,String> map = new HashMap<String,String>();
    map.put("aa", "美美");
    map.put("bb", "小凤");
    map.put("cc", "芙蓉");
    request.setAttribute("map", map);
%>
<c:forEach var="entry" items="${ map }">
    ${ entry.key } -- ${ entry.value }
</c:forEach>

<h4>遍历对象的集合</h4>
<%
    List<User> uList = new ArrayList<User>();
    uList.add(new User("美美","123"));
    uList.add(new User("小凤","234"));
    uList.add(new User("芙蓉","345"));
    request.setAttribute("uList", uList);
%>
<c:forEach var="user" items="${ uList }">
    ${ user.username } -- ${ user.password }
</c:forEach>


<h4>迭代数据</h4>
<h4>迭代从1到10</h4>
<c:forEach var="i" begin="1" end="10" step="2">
    ${ i }
</c:forEach>


<h4>计算从1加到100的和</h4>
<c:set var="sum" value="0" scope="page"></c:set>
<c:forEach var="i" begin="1" end="100" step="1">
    <c:set var="sum" value="${ sum + i }"></c:set>
</c:forEach>
${ sum }

<h4>遍历10到100的偶数，每到第3个数，显示红色</h4>
<c:forEach var="i" begin="10" end="100" step="2" varStatus="status">
    <c:choose>
        <c:when test="${ status.first }">
            <font color="blue">${ i }</font>
        </c:when>
        <c:when test="${ status.count % 3 eq 0 }">
```

```
                    <font color="red">${ i }</font>
                </c:when>
                <c:otherwise>
                    ${ i }
                </c:otherwise>
            </c:choose>
        </c:forEach>
```

```
* <c:param>      传递参数
    * 属性
        name     : 参数名称
        value    : 参数的值

* <c:import>     包含页面
    * url                : 引入页面的地址
    * context    : 虚拟路径
    * var                : 引入页面保存到属性中 ,注意调用 ${i} 时不能在标签<c:import ></c:import>体内
    * scope              : 域的范围

    * 代码
        <c:import url="/jstl/choose.jsp" context="/day13" var="i" scope="page">
            <c:param name="username" value="meimei"></c:param>
        </c:import>
        ${ i }

* <c:url>
    <c:url>标签用于在JSP页面中构造一个URL地址，其主要目的是实现URL重写。
    URL重写就是将会话标识号以参数形式附加在URL地址后面
    http://localhost/day12/demo?jsessionid=xxxxxxxxxxxxxxxxxx;

    * 属性
        * var            : 声明属性
        * value          : 地址
        * scope          : 域范围
        * context        : 虚拟路径

    * 代码
        <c:url var="i" value="/jstl/choose.jsp" scope="request" context="/day13">
            <c:param name="username" value="xiaofeng"></c:param>
        </c:url>

        <a href="${ i }">choose</a>

        链接地址: /day13/jstl/choose.jsp?username=xiaofeng

* <c:redirect>重定向
    * 属性
        * url            : 重定向的地址
        * context        : 虚拟路径
    * 代码
        <c:redirect url="/jstl/choose.jsp" context="/day13">
            <c:param name="username" value="furong"></c:param>
        </c:redirect>

* <c:forTokens />分隔字符串（了解）
    <h4>分隔字符串</h4>
    <c:set var="i" value="aa,bb,cc" scope="page"></c:set>
    <c:forTokens items="${i }" delims="," var="x">
        ${ x }
    </c:forTokens>
```

自定义 JSTL 标签

什么是自定义标签

尽管JSP本身，以及第三方提供了很多标签库，但因为业务需要，我们有时还是需要自定义标签。因为JSP页面中不可以存在Java代码，所以我们需要自定义标签！

标签的真身

标签可以是一个对象。其实在页面中使用的标签就是对一个对象的方法调用！ 标签： 标签处理类：都有自己的标签处理类！所有标签处理类都必须去实现Tag或SimpleTag接口； TLD（Tag Library Description）：一个TLD文件中可以部署多个标签！JSP会通过TLD文件找到标签！

快速入门，基本自定义标签

SimpleTag

1. SimpleTag是什么 标签处理类必须实现JspTag接口，而JspTag接口有两个子接口：Tag和SimpleTag。更加确切的说：标签处理类必须实现Tag或SimpleTag接口。JSP2.0之后，SUN提供了SimpleTag接口，通过SimpleTag接口来实现标签处理类要比Tag这种传统方式方便很多，所以现在我们可以大声与Tag说再见了。 SimpleTag接口内容如下：（后台调用顺序，从上至下）

- void setJspContext(JspContext pc) ： 设置PageContext(JspContext 是PageContext的子类)
- void setParent(JspTag parent) ： 设置父标签
- void setJspBody(JspFragment jspBody) ： 设置标签体对象；
- void doTag() 标签执行方法；
- JspTag getParent() 获取父标签；

2. 自定义标签类 继承 SimpleTagSupport

```
public class HelloTag extends SimpleTagSupport{
        private PageContext pageContext;
        public void doTag() throws JspException, IOException {
        // 主要执行方法，标签的作用取决于改变方法
                pageContext.getOut().write("<p>Hello SimpleTag!</p>");
        }
        public void setJspContext(JspContext pc) {
                this.pageContext = (PageContext) pc;
        }

        public void setParent(JspTag parent) {}
        public JspTag getParent() {return null;}

        public void setJspBody(JspFragment jspBody) {}
}
```

3. 创建.tld 文件配置

```
    <!-- 配置自定义标签 -->
    <tag>
        <!-- 配置标签名称 -->
        <name>print</name>
        <!-- 配置标签的类 -->
        <tag-class>cn.itcast.tag.TagDemo1</tag-class>
        <!-- 配置标签主体 -->
        <body-content>empty</body-content>
    </tag>
```

4. 在JSP页面上，引入标签库 <%@ taglib uri="http://www.itcast.cn/1110/myc" prefix="myc" %>

5. 使用 ${myc:print("hello")}

有标签体的标签

- 获取标签主体对象

```
    JspFragment jf = getJspBody();
    jf.invoke(null);
```

1. 标签类

```
public class MyTagPrintBody extends SimpleTagSupport{
        private PageContext pageContext;
        @Override
        public void doTag() throws JspException, IOException {
                JspFragment jspBody  = getJspBody();  // 获取标签体对象
                jspBody.invoke(pageContext.getOut());
                //  也可以写成   jspBody.invoke(null);
        }
        @Override
        public void setJspContext(JspContext pc) {
                pageContext=(PageContext) pc;
        }
}
```

2. 配置

```xml
<!-- 配置自定义标签 -->
    <tag>
    <!-- 配置标签名称 -->
    <name>out</name>
    <!-- 配置标签的类 -->
    <tag-class>cn.itcast.tag.TagDemo2</tag-class>
    <!-- 配置标签主体 -->
    <body-content>scriptless</body-content>
    </tag>
```

3. 使用 <myc:out>输出该内容</myc:out>

带有属性的标签

1. 编写类，继承SimpleTagSupport。

- 重写doTag()
- 编写一个属性，属性必须和标签中的属性是相同
- 提供set方法

例:

```
public class MyTagAttribute extends SimpleTagSupport {
        private String first;
        private String second;
        private PageContext pageContext;

        public void setFirst(String first) {
                this.first = first;
        }
        public void setSecond(String second) {
                this.second = second;
        }
        @Override
        public void doTag() throws JspException, IOException {
                JspWriter jspWriter = pageContext.getOut();
                int res = first.compareTo(second);
                if(res<0){
                        jspWriter.print(first+"<"+second);
                }else if(res>0){
                        jspWriter.print(first+">"+second);
                }else{
                        jspWriter.print(first+"=="+second);
                }
        }
        @Override
        public void setJspContext(JspContext pc) {
                this.pageContext = (PageContext) pc;
        }
}
```

2. .tld配置

```
<!-- 配置自定义标签 -->
    <tag>
    <!-- 配置标签名称 -->
    <name>myc</name>
    <!-- 配置标签的类 -->
    <tag-class>cn.itcast.tag.TagDemo3</tag-class>
    <!-- 配置标签主体 -->
    <body-content>scriptless</body-content>
    <!-- 配置属性 -->
    <attribute>
        <!-- 配置属性名称 -->
        <name>compare</name>
        <!-- 属性是否是必须的 -->
        <required>true</required>
        <!-- 是否支持EL表达式 -->
        <rtexprvalue>true</rtexprvalue>
        <!-- 属性的类型 -->
        <type>String</type>
    </attribute>
```

```
    </tag>
```

3. 使用

```
        <myc:compare first="20" second="30" /> <!-- 20<30 -->
        <myc:compare first="30" second="30" /> <!-- 30==30 -->
        <myc:compare first="40" second="30" /> <!-- 40>30 -->
```

扩展

不执行标签下面的页面内容

我们知道，在使用Tag接口来编写标签时，可以跳过标签下面的JSP页面内容。在SimpleTag中也是可以的，这需要在doTag()方法中抛出SkipPageException。
SkipPageException是JspException的子类，当doTag()方法抛出该异常时，JSP真身不会打印异常信息，而是跳转页面内容！

```
public class SkipTag extends SimpleTagSupport {
    public void doTag() throws JspException, IOException {
        this.getJspContext().getOut().print("<h1>只能看到我！</h1>");
        throw new SkipPageException();
    }
}
    <tag>
        <name>skip</name>
        <tag-class>cn.itcast.simpletags.SkipTag</tag-class>
        <body-content>empty</body-content>
    </tag>
<itcast:skip/>
<h1>看不见我！</h1>
```

7 带有属性的标签
≒ 在处理类中添加属性，以及getter/setter方法；
≒ 在TLD中部属相关属性。

```
public class DemoTag extends SimpleTagSupport {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void doTag() throws JspException, IOException {
        this.getJspContext().getOut().print("hello " + name);
    }
}
    <tag>
        <name>demo</name>
        <tag-class>cn.itcast.simpletags.DemoTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <name>name</name>
            <required>true</required>
            <rtexprvalue>false</rtexprvalue>
        </attribute>
    </tag>
<%
pageContext.setAttribute("name", "liSi");
String name = "wangWu";
%>
    <itcast:demo name="zhangSan"/><br/>
    <itcast:demo name="${name }"/><br/>
    <itcast:demo name="<%=name %>"/>
```

标签练习

1 referer（防盗链）
功能：当访问使用了防盗链的页面时，标签会获取请求头信息中的Referer，查看是否与标签中给定的前缀相同，如果不同，那么重定向到指定的页面。
防盗链标签两个属性：
≒ site：请求头中的Referer如果不是site前缀，那么就是盗链；

≒　　path：当发现为盗链时，重定向到path指定的位置。

```
public class RefererTag extends SimpleTagSupport {
    private String site;// 允许的位置，例如：http://localhost:8080/为前缀的URL是允许的。
    private String path;// 一旦出现盗链，跳转到哪里
    public void setSite(String site) {
        this.site = site;
    }
    public void setPath(String path) {
        this.path = path;
    }
    public void doTag() throws JspException, IOException {
        PageContext pageContext = (PageContext) this.getJspContext();
        HttpServletRequest request = (HttpServletRequest) pageContext.getRequest();
        String referer = request.getHeader("referer");
        if(referer == null || !referer.startsWith(site)) {
            HttpServletResponse response = (HttpServletResponse) pageContext.getResponse();
            response.sendRedirect(request.getContextPath() + path);
            throw new SkipPageException();
        }
    }
}
```

2　if标签

功能：模仿JSTL中的<c:if>标签

```
public class IfTag extends SimpleTagSupport {
    private boolean test;

    public void setTest(boolean test) {
        this.test = test;
    }

    @Override
    public void doTag() throws JspException, IOException {
        if(test) {
            this.getJspBody().invoke(null);
        }
    }
}
```