

Java Server Pages

--- Web Base Course , Module 4

Royallin High-End IT Training , JSP

Summary

- Introduction to JSP
- Scripting Elements
- Implicit Objects
- Standard Actions
- Expression Language
- Custom Tags
- Java Standard Tag Lib

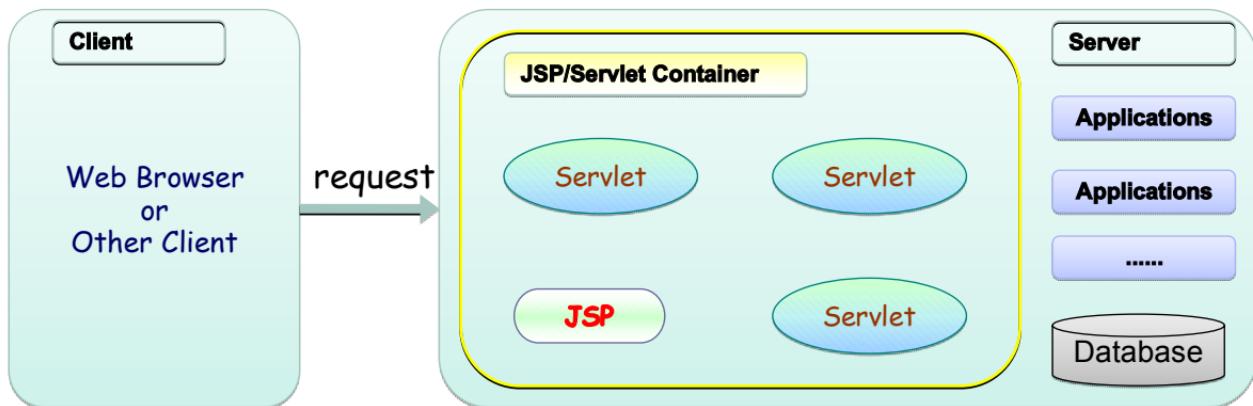
Why use JSPs ?

JSP Container

JSP在Web应用中的地位

常用的web容器

- Apache Tomcat : Apache 提供
- JBoss : Red Hat 提供
- BEA weblogic : Oracle 提供



JSP Container

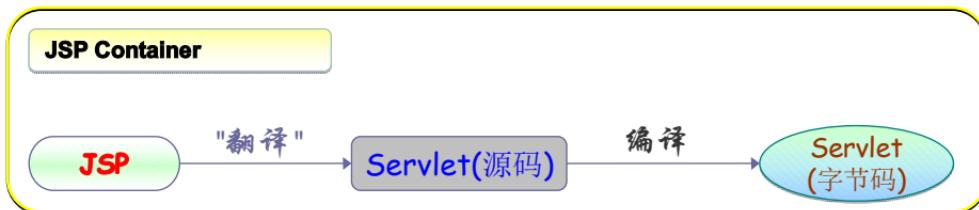
JSP与Servlet容器

Servlet 容器

- ◆ Servlet 是一种运行在服务器端的 Java 小程序
- ◆ 为 Servlet 提供运行环境的程序，视为 Servlet 容器

JSP容器

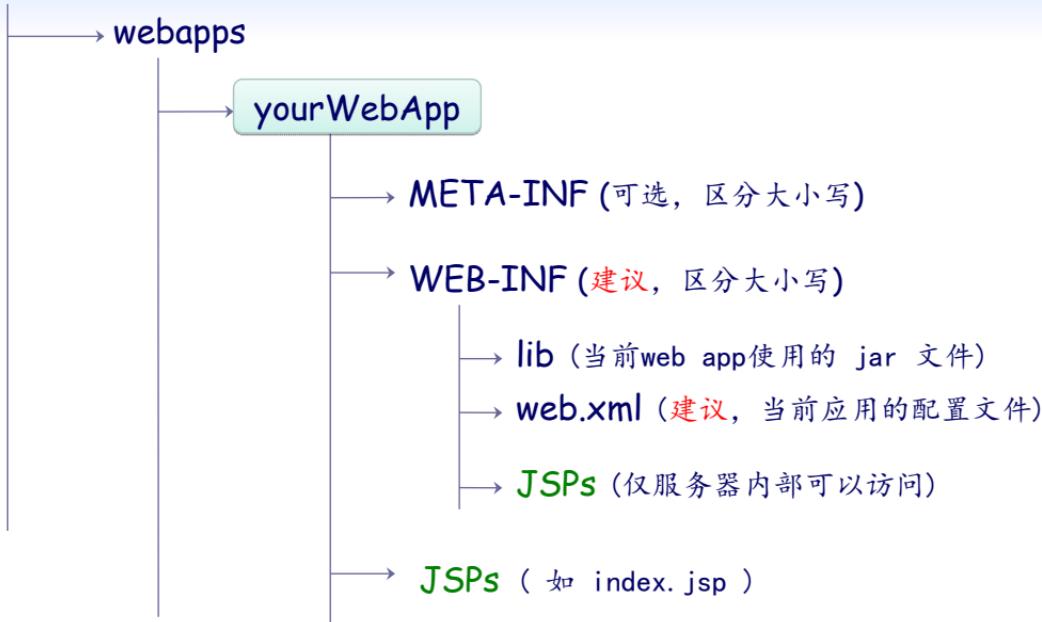
- ◆ 后续课程会看到，JSP 是一种特殊的 Servlet
 - JSP 必须由程序先"翻译"成 Java 程序(Servlet源码)，之后编译成字节码文件
- ◆ JSP 容器需要动态"翻译" JSP 页面、编译 Servlet 源码
 - 这是JSP 容器与 Servlet 容器的一个明显区别



JSP Container

- 含有JSP页面的web应用在JSP容器中的目录层次

Tomcat_Home



The First JSP example

■ 开发第一个 JSP 页面

- 在 Tomcat_Home/webapps/yourApp 中创建一个 jsp 页面
- 内容如下

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Hello</title>
        <meta charset="UTF-8" />
    </head>
    <body>
        Hello , This is my first JavaServer Page !
        <br>
        <%= new java.util.Date() %>
    </body>
</html>
```

The First JSP example

- 访问第一个JSP页面

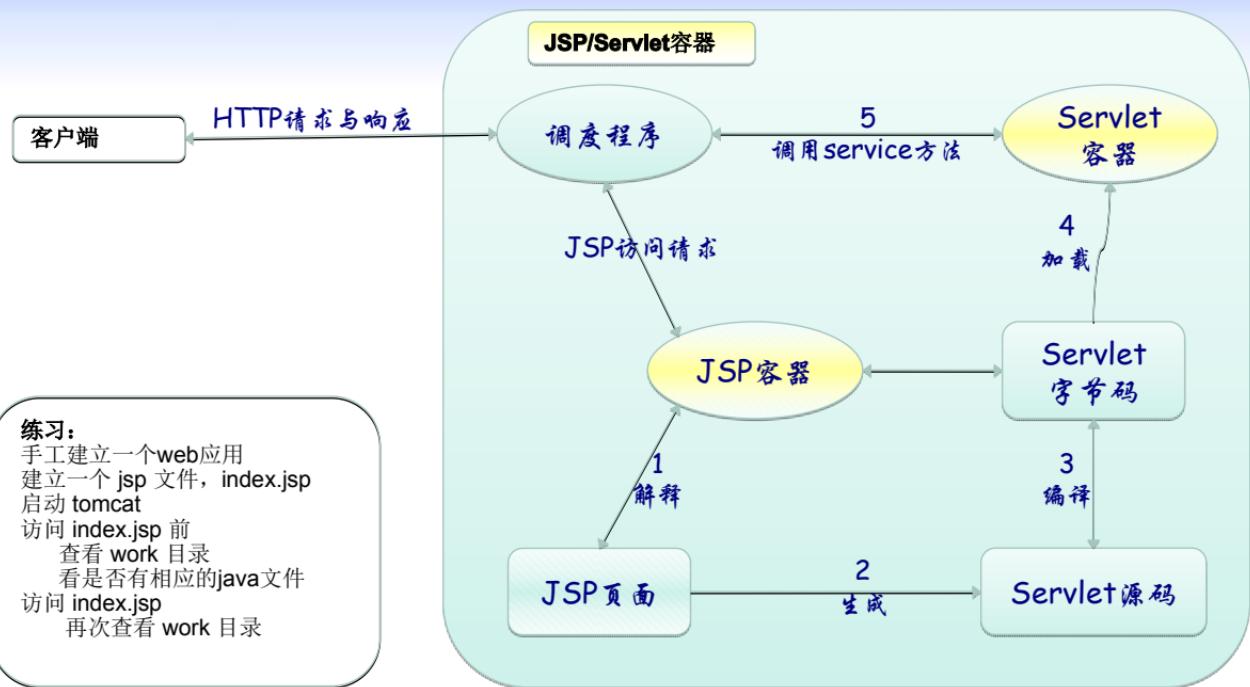
■ 第一次访问JSP页面时

- JSP 容器解释 JSP 页面生成 Servlet源文件(.java 文件)
 - ◆ . java 文件被存放在 Tomcat_Home/work 目录下
 - work/Catalina/localhost/yourappname/org/apache/jsp/
 - index.jsp 对应的 .java 文件，名字叫做 index_jsp.java
- 编译生成的 Servlet (.class 文件)
 - ◆ . class 文件跟 .java 放在同一个目录底下
 - index.jsp 对应的 .class 文件，名字叫做 index_jsp.class
- Servlet容器将 Servlet 加载到 JVM
- 调用 Servlet 的 service 方法

■ 第一次以后

- 先检查JVM中是否有要访问的Servlet，若没有，加载之
- 调用相应 Servlet 的 service 方法

JSP Lifecycle



练习：
 手工建立一个web应用
 建立一个 jsp 文件，index.jsp
 启动 tomcat
 访问 index.jsp 前
 查看 work 目录
 看是否有相应的java文件
 访问 index.jsp
 再次查看 work 目录

JSP Lifecycle

Directives

■ What is Directive ?

- Directive , 指令, 用来声明 JSP 页面的一些属性
 - ◆ 这些页面属性如页面的类型, 字符编码等
 - ◆ 指令可以理解成写在页面上, 告知JSP容器如何翻译当期页面
 - ◆ 因此, 指令是写在 JSP 页面上, 但是是向 JSP 容器"下达"的指令
- JSP 指令的格式

```
<%@ directive { attribute="value" }* %>
```

- ◆ 其中
 - directive 指指令名称, 常用指令有 page 、 include 、 taglib 等
 - 而 attribute="value" 则是属性和属性值, { }* 表示可以有 0 到 n 个属性
- 同一个页面上, 同一个指令可以出现多次, 属性不同即可

```
<%@ page pageEncoding="UTF-8" %>  
<%@ page import="java.util.*" %>
```

Directives

page Directive

- page 指令是最常用的指令，用来声明 JSP 页面的属性的
- page 指令常用属性及作用如下表

属性名称	取值范围	描述
language	java	指明解释该 JSP 文件时采用的语言。一般为 Java 语言。默认为 Java
extends	任何类的全名	指明编译该 JSP 文件时继承哪个类。JSP 为 Servlet，因此当指明继承普通类时需要实现 Servlet 的 init, destroy 等方法
import	任何包名, 类名	引入该 JSP 中用到的类、包等。import 是唯一可以声明多次的 page 指令属性。一个 import 属性可以引用多个类, 中间用英文逗号隔开, 如<%@ page import="java.util.List, java.util.ArrayList" %>。JSP 中下面四个包里的类可以直接使用: <code>java.lang.*</code> , <code>javax.servlet.*</code> , <code>javax.servlet.jsp.*</code> , <code>javax.servlet.http.*</code>
session	true, false	指明该 JSP 内是否内置 Session 对象。如果为 true, 则内置 Session 对象, 可直接使用。否则不内置 Session 对象。默认为 true
autoFlush	true, false	是否运行缓存。如果为 true, 则使用 <code>out.println()</code> 等方法输出的字符串并不是立刻到达客户端服务器的, 而是暂时存在缓存里, 缓存满或者程序执行完毕或者执行 <code>out.flush()</code> 操作时才到客户端。默认为 true
buffer	none 或者 数字+kB	指定缓存大小。当 autoFlush 设为 true 时有效, 例如: <%@ page buffer="10kb" %>
isThreadSafe	true, false	指定是否线程安全。如果为 true, 则运行多个线程同时运行该 JSP 程序, 否则只运行一个线程运行, 其余线程等待。默认为 false
isErrorPage	true, false	指定该页面是否为错误处理页面。如果为 true, 则该 JSP 内置有一个 <code>Exception</code> 对象 <code>exception</code> , 可直接使用, 否则没有。默认为 false
errorCode	某个 JSP 页面的相对路径	指明一个错误显示页面, 如果该 JSP 程序抛出了一个未捕捉的异常, 则转到 <code>errorCode</code> 指定的页面。 <code>errorCode</code> 指定的页面通常 <code>isErrorPage</code> 属性为 true, 且内置的 <code>exception</code> 对象为未捕捉的异常
contentType	有效的文档类型。	客户端浏览器根据该属性判断文档类型, 例如: HTML 格式为 <code>text/html</code> 纯文本格式为 <code>text/plain</code> JPG 图像为 <code>image/jpeg</code> GIF 图像为 <code>image/gif</code> WORD 文档为 <code>application/msword</code>
info	任意字符串	指明 JSP 的信息。该信息可以通过 <code>Servlet.getServletInfo()</code> 方法获取到
trimDirectiveWhitespaces	true, false	是否去掉指令前后的空白字符。默认为 false

■ include Directive

- 用来实现 include 效果，即本应用程序中包含其他JSP或HTML
- include 指令比较简单，只有一种格式

```
<%@ include file="relativeURL" %>
```

- ◆ 其中 relativeURL 可以是 JSP 页面或者 HTML 页面
- include 一般用于页面内容的区域化
 - ◆ 比如，一个站点需要有统一的 head 和 foot，可以使用 include 实现

■ taglib Directive

- JSP支持标签技术，用于实现表示层的代码重用
- 使用标签，必须先声明标签库以及标签前缀
- taglib 指令用来指明 JSP 页面内使用的JSP标签库
- taglib 指令有两个属性
 - ◆ uri 为类库的地址
 - ◆ prefix 为标签的前缀

```
<%@ taglib uri="/struts-tags" prefix="s" %>
```

Scope

What is script ?

Comments

Declarations

Expressions

Scriptlets

What is Implicit Object

- Concept (概念)

request & response

request

- 本质上是 javax.servlet.ServletRequest 类型的对象
 - ◆ 与之相联系的类型是 javax. servlet. http. HttpServletRequest
- 该类型的对象封装了客户某一次的所有的请求数据
 - ◆ 使用 javax. servlet. ServletRequest 中的相关方法可以获取这些请求数据
- 其有效的作用范围仅仅是当前请求(request)
 - ◆ 即当前的 request 对象只在当前请求内起作用
 - ◆ 除非是使用了 RequestDispatcher 的 forward 或者 include

response

- 本质上是 javax.servlet.ServletResponse 类型的对象
 - ◆ 与之相联系的类型是 javax. servlet. http. HttpServletResponse
- 该类型的对象封装了服务器对客户某次请求的响应结果
 - ◆ 其中的输出是存放到缓冲的，因此可以设置响应的HTTP状态和响应头
- 其有效范围仅仅是当前响应对应的页面(page)

request & response

■ Create request and response

- 二者都由 jsp/servlet 容器创建
- 通过 `_jspService()` 方法传入
 - ◆ `_jspService(HttpServletRequest request , HttpServletResponse response)`
 - ◆ 任何 jsp 页面最终对应的 servlet 中都包含 `_jspService` 方法

■ Scope

- 从代码角度看
 - ◆ `request` 和 `response` 仅仅作用于 `_jspService` 方法内
- 实际上
 - ◆ `request` 对象仅仅在本次请求内有效
 - 当在当前 jsp 中执行了 `include` 和 `forward` 操作时特殊对待
 - ◆ `response` 对象则仅仅在它所对应的响应页面有效

■ 用于向客户端输出响应数据

- 在 JSP 中 out 的类型是 javax.servlet.jsp.JspWriter
 - ◆ 这个类型是一个类似于 java.io.PrintWriter 的优化版本
 - ◆ 它继承了 java.io.Writer，为了使 response 有效，它增加了缓冲功能
- 在 jsp 页面中可以使用 page 指令设置是否使用缓冲和缓冲大小
 - ◆ <%@ page ... buffer="8kb" autoFlush="true" %> : 使用缓冲，并自动刷新
 - ◆ <%@ page ... buffer="none" autoFlush="true" %> : 不使用缓冲

■ 代表客户端与服务器的一次会话

- 本质上是 javax.servlet.http.HttpSession 类型的对象
 - ◆ HttpSession session = null ;
 - ◆ _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
 - ◆ session = pageContext.getSession() ;
- 在 jsp 中 session 对象是自动创建好的，因此可以直接使用
 - ◆ 在某个客户端首次访问 jsp 页面时，容器就创建好了 session
 - ◆ 如果使用 page 指令关闭了 session，则本页面不能再使用 session
 - <%@ page ... session="false" %> : 这里把 session 设置为 false 即为不使用 session

■ 指代当前 web 应用程序

- 本质上是 javax.servlet.ServletContext 类型的对象
- 该对象是一个及其重要的对象
 - ◆ 在容器加载某个 web 应用时，即为该 web 应用创建一个 application 对象
 - ◆ 该对象一经创建，将一直保持下去，直到容器关闭或卸载该 web 应用
- 在 servlet 中获取该对象的引用
 - ◆ GenericServlet.getServletContext()
 - ◆ HttpServlet.getServletContext()
 - ◆ ServletConfig.getServletContext()
 - ◆ HttpSession.getServletContext()
- 在 jsp 中获取该对象的引用
 - ◆ 直接使用 application 对象即可，jsp 中通过如下方式获取
 - ServletContext application = pageContext.getServletContext();

■ 用于设置当前 jsp 页面

- 本质上是 javax.servlet.ServletConfig 类型的对象
 - ◆ `ServletConfig config = pageContext.getServletConfig();`
- 使用 config 可以获取该 jsp 的初始化参数
 - ◆ `config.getInitParameter(name);`
- 也可以使用该对象获得 ServletContext
 - ◆ `config.getServletContext();`

pageContext & page

page

- 代表当前 jsp 页面本身，等同于 this 关键字
- 它仅仅存在于 jsp 文件中，转换后的 java 文件中不包含 page

pageContext

- 代表当前 jsp 页面的上下文
- 本质上是 javax.servlet.jsp.PageContext 类型的对象
 - ◆ 该类型是 jsp 中所新增的类型，在 servlet 中没有述及
- 主要作用是管理对于属于 jsp 中特殊可见部分中已命名对象的访问
 - ◆ pageContext.getServletContext()
 - ◆ pageContext.getSession()
 - ◆ pageContext.getRequest()
 - ◆

■ Intention (作用)

- 对应 jsp 页面上的异常，类型为 java.lang.Throwable

■ Use

- 定义一个专门用于处理异常的页面 (比如: exception.jsp)
 - ◆ 必须使用 page 指令指该页面定为异常处理页面，才能使用 exception 对象
 - ◆ <%@ page ... isErrorPage="true" %> : isErrorPage 的默认值是 false
- 其它页面使用 page 指令指定使用上面的页面作为异常处理页面
 - ◆ <%@ page ... errorPage="exception.jsp" %> : errorPage 的默认值是 null
- 当访问某个页面时引发了异常，则自动定位到错误处理页面
 - ◆ 在该页面上可以显示异常信息或者用其它方式处理异常

What are JSP Actions ?

■ Concept (概念)

- Action 是一个 JSP 元素
 - ◆ 可以操作内置对象、其他服务器对象
 - ◆ 或者定义一个新的脚本变量，或者完成转发、包含等行为
- 利用 XML 语法格式的标记来控制 Servlet 引擎的行为
 - ◆ 遵循 XML 语法，必须有开始标记、主体、结束标记
 - 如果主体是空的，可以使用空标记语法
 - ◆ Standard Action 必须使用 `jsp:` 做前缀
- Action 可以分成 标准动作 和 自定义动作

■ General Syntax (通用语法)

```
<jsp:actionName attribute="attrValue" > body </jsp:actionName>
```

■ Meaning (意义)

- 简化 JSP 页面，实现 Java 代码与页面分离

Actions for JSP 2.0

元素名称	简单描述	元素名称	简单描述
<jsp:useBean>	存取JavaBean	<jsp:declaration>	主要用在JSP Document之中
<jsp:setProperty>	存取JavaBean	<jsp:scriptlet>	主要用在JSP Document之中
<jsp:getProperty>	存取JavaBean	<jsp:expression>	主要用在JSP Document之中
<jsp:include>	包含静态或动态的网页文件	<jsp:text>	主要用在JSP Document之中
<jsp:forward>	网页转向	<jsp:output>	主要用在JSP Document之中
<jsp:param>	提供key/value的资讯	<jsp:attribute>	定义XML元素属性或标签属性值
<jsp:plugin>	在浏览器中播放或显示一个物件	<jsp:body>	定义XML元素标签本体内容
<jsp:params>	通常和plugin搭配使用	<jsp:element>	动态定义XML元素标签值
<jsp:fallback>	通常和plugin搭配使用	<jsp:incoke>	主要用在Tag File之中
<jsp:root>	主要用在JSP Document之中	<jsp:doBody>	主要用在Tag File之中

The Include Action

Intention (作用)

- 允许您包括在 JSP 文件中的静态或动态资源
 - 静态资源 (static resource)
 - 把静态资源的内容插入到调用它的 JSP 中
 - 动态资源 (dynamic resource)
 - 请求会被发送到被包含的动态资源中
 - 动态资源执行完毕，把返回的 response 包含到调用它的 JSP 中

Syntax (语法)

```
<jsp:include page="urlSpec" flush="true | false" />  
or  
<jsp:include page="urlSpec" flush="true | false" >  
  { <jsp:param ... /> }*  
</jsp:include> <%--the default value of flush is "false" --%>
```

The Include Action

Attributes

- **page** 用于指定所要包含的资源相对URL
 - ◆ 可以使用相对路径
 - `page="includeResource"` : 被包含的资源与当前页面在同一层次
 - `page="../includeResource"` : 被包含资源在当前页面的父层次中
 - ◆ 也可以使用"绝对路径"
 - `page="/includeResource"` : 被包含资源在当前应用的根目录下
 - 这个路径是相对于当前web应用的根目录的, 本质上还是相对路径
 - 因为在容器中绝对路径是从 容器的根目录开始算起的(当前web应用的上一层目录)
 - ◆ 被包含的资源可以是 `jsp` 也可以是 `servlet` 或者 `html` 等
- **flush** 用于设置当缓冲区用尽时, 是否清空
 - ◆ 默认值是 `false` , 在 `jsp 1.2` 前, 必须把这个值设置为 `true`

The Include Action

Example (举例)

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
  <head><title>Include Action Sample</title></head>
  <body>
    <jsp:include page= "banner.html"/>
    <jsp:include page="date.jsp">
      <jsp:param name="user" value="George"/>
    </jsp:include>
  </body>
</html>
```

inclActSample.jsp

The Include Action

Example (举例)

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@page import="java.util.*"%>
<h2><center>Hello, <%=request.getParameter("user")%>,
<%
    Calendar cal = Calendar.getInstance();
    if( cal.get(Calendar.AM_PM) == Calendar.AM ) {
        out.println("good morning!");
    } else {
        out.println("good afternoon!");
    }
%>
</center></h2>
```

date.jsp

The Forward Action

Intention (作用)

- 把控制权从一个 JSP 页面转发到另一个 web 组件
 - ◆ 如果页面输出缓冲
 - 在转发之前清除缓冲区
 - 缓冲区已被刷新，企图将请求转发将导致 IllegalStateException

Syntax (语法)

```
<jsp:forward page="urlSpec" />  
or  
<jsp:forward>  
  <jsp:attribute name="page">urlSpec</jsp:attribute>  
</jsp:forward>  
or  
<jsp:forward page="urlSpec">  
  { <jsp:param ... /> }*  
</jsp:forward>
```

The Forward Action

Attributes

- page 指定将要转发到的目标 URL
 - ◆ 使用相对路径

Sub Elements

- <jsp:attribute> : 设置参数, 比如设置 page
 - ◆ 相当于在 page 属性中设置 url
- <jsp:param> : 为目标 URL 传递参数
 - ◆ name : 指定所传递的参数的名称
 - ◆ value : 指定所传递的参数的参数值

The Forward Action

Contrast (对比)

```
<jsp:forward page="/shoppingcar/add">  
    <jsp:param name="pid" value="12" />  
</jsp:forward>
```

```
.....  
request.getRequestDispatcher( "/shoppingcar/add?pid=12" )  
    .forward( request, response );  
.....
```

转发过去以后，接收传来的参数

```
String productid = request.getParameter( "pid" );
```

The Parameter Action

■ Intention (作用)

- 提供 key/value 形式的信息
 - ◆ 使用在 `jsp:include` 、 `jsp:forward` 、 `jsp:params` 等元素中
- 主要用途是用来传递参数

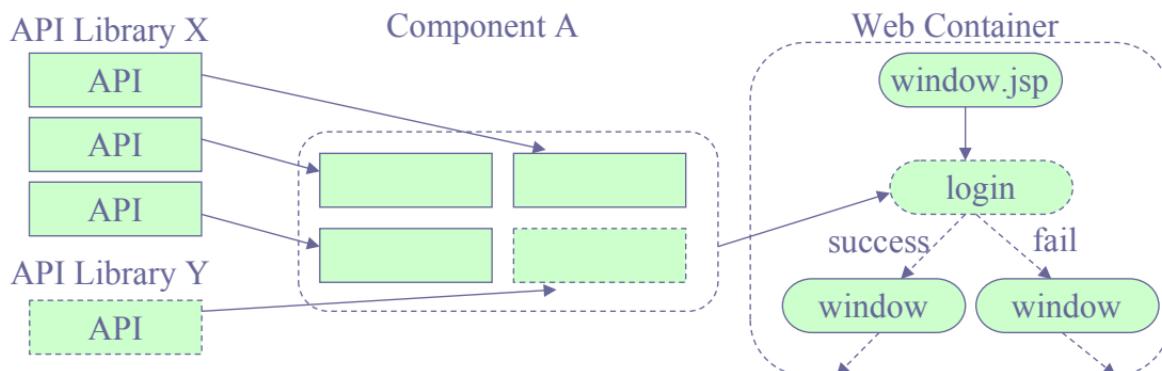
■ Syntax (语法)

```
<jsp:param name="paramName" value="paramValue"/>
```

The useBean Action

Software Components (软件组件)

- 把可用的、底层的 API 集合组合成可重复使用的程序
 - 按照某种规范进行组合
- 组合后的程序，可以处理更高级的任务
- 常见的组件技术
 - Java Beans 、Enterprise JavaBeans (EJB) 、.....



The useBean Action

■ What are JavaBeans

■ Concept

- ◆ 按照某种标准格式编写的Java类
- ◆ 全面的介绍和学习 JavaBeans 超出了JSP的范围
- ◆ 如有兴趣，可以系统学习 Enterprise Java Beans (EJB)

■ Three meanings

- ◆ First : JavaBeans 是一种规范
 - 是一种在Java (包括JSP) 中使用可重复使用的Java组件的技术规范
- ◆ Second : JavaBeans是一个Java的类
 - 一般来说，这样的 Java 类将对应于一个独立的 .java 文件
 - 在绝大多数情况下，这应该是一个 public 类型的类
- ◆ Third : 有时称JavaBeans的实例为JavaBeans
 - 当 JavaBeans 这样的一个 Java 类在具体的 Java 程序中被实例之后
 - 我们有时也会将这样的一个JavaBeans的实例称之为JavaBeans

The useBean Action

JavaBean 的编写要求

- Bean 类必须拥有一个无参构造方法(就是默认的那个)
 - ◆ 显式地定义这样一个构造, 或者省略所有的构造
 - ◆ JSP元素创建Bean时, 会调用默认的构造函数
 - 实际应用中, 常用 Servlet 创建 Bean , 此时, 不必一定有无参构造
- Bean 类不应该有公开的实例变量(或叫字段、属性)
 - ◆ 所有的属性被私有化(封装起来)
- 持续性的值应该通过 getXxx 和 setXxx 方法来访问
 - ◆ 对于私有的属性需要提供对外访问的方法: get 和 set

暂时不对 Java Beans 做全面研究

如前所述, 关于JavaBean, 我们暂时不能做详细学习
就 Java Bean 在 JSP 中的使用来说, 了解前面 3 点足矣

The useBean Action

■ Intention (作用)

- 在 JSP 页面中创建或者查找一个 Bean 实例
 - ◆ 可以指定查找的范围, 比如 page 、 request 、 session 、 application
- 也可以指定该 Bean 实例的类型 (class 或 type)

■ Syntax (语法)

```
<jsp:useBean id="name"  
    scope="page | request | session | application"  
    typeSpec  
/>
```

- 其中 typeSpec 可以写成如下形式之一:
 - ◆ class = "className"
 - ◆ class = "className" type = "typeName"
 - ◆ beanName = "beanName" type = "typeName"
 - ◆ type = "typeName"

The useBean Action

Attributes

- **id="name"**
 - ◆ 用以指定 Java 对象的实例名或引用对象的脚本变量的名称
 - ◆ 通过id指定的名称来分辨不同的Bean
 - ◆ 该名称严格区分大小写(相当于Java中的变量的名称)
- **scope="page | request | session | application"**
 - ◆ 该属性用以确定 Bean 存在的范围以及 id 变量名的有效范围
 - ◆ page : 当前JSP文件及其所有静态包含文件中可以使用Bean
 - 这个是默认值
 - ◆ request : 在任何执行相同请求的JSP文件中可以使用Bean
 - 可以通过request对象访问Bean : `request.getAttribute(name)`
 - ◆ session : 整个会话(Session) 范围内都可以使用Bean
 - 可以通过session对象访问Bean : `session.getAttribute(name)`
 - 注意, 首次创建Bean的JSP文件中必须在page指令中指定 `session=True` !!!
 - ◆ application : 整个应用程序(Application) 内都可以使用Bean
 - 使用application 对象获取Bean : `application.getAttribute(name)`

The useBean Action

■ About "typeSpec"

■ class="className"

- ◆ 使用 new 关键字 和 构造器构造一个实例
- ◆ 这个类必须是非抽象的，必须有公共的、无参构造
- ◆ 类名必须是 包名 + 类名，严格区分大小写

■ beanName="beanName" type="typeName"

- ◆ 使用 instantiate 方法从一个 class 中实例化一个Bean
 - instantiate方法在 java.beans.Beans 中
- ◆ beanName可以是 package 或 class ，也可以是表达式
 - package和class都是大小写敏感的
- ◆ 同时还可以指定 Bean 的类型

■ type="typeName"

- ◆ type可以指定一个类，也可以是一个父类，或者是一个接口
- ◆ 如果没有使用class或beanName指定type， Bean将不会被实例化

The getProperty Action

■ Intention (作用)

- 获取Bean的值，并将其转化为一个字符串插入到输出页面中

■ Syntax (语法)

```
<jsp:getProperty name="name" property="propertyName"/>
```

■ About attribute

- ◆ name="name" : 必选属性，相应的Bean的名字
- ◆ property="propertyName" : 必选属性，其值为相应的Bean内的属性名

■ 使用时需注意

- ◆ 使用该动作之前，相应的bean必须已经创建
- ◆ 不能使用<jsp:getProperty>来检索一个已经被索引了的属性
- ◆ 能够和JavaBeans组件一起使用<jsp:getProperty>
- ◆ 不能与 Enterprise Java Bean (EJB) 一起使用

The setProperty Action

■ Intention (作用)

- 用以设置 Bean 中属性的值

■ Syntax (语法)

```
<jsp:setProperty name="objName" property-expression />
```

- property-expression 可以写成：

- ◆ property="*"
- ◆ property="propertyName"
- ◆ property="propertyName" param="parameterName"
- ◆ property="propertyName" value="propertyValue"

The setProperty Action

■ About property-expression

■ property="*"

- ◆ 通过用户输入的所有数据来匹配 Bean 中的属性
- ◆ 在Bean中，属性的名字、类型必须跟request对象中的参数名称、类型一致
- ◆ 从客户端上传到服务器的参数值一般都为字符串类型
 - 它们需要转换成其他的类型（使用包装类的 valueOf 方法）
- ◆ 如果request对象的参数值中有空值
 - 那么对应的Bean属性将不会被设置任何值
- ◆ 如果Bean中有一个属性没有与之对应的request参数值
 - 那么这个属性同样也不会被设定
- ◆ 使用 property="*"
 - Bean的属性没有必要按照 html 表单的顺序排序
 - 同样，html 表单的顺序也没有必要非得按照 Bean 中属性的顺序排序

The setProperty Action

■ About property-expression

■ property="propertyName"

- ◆ 使用一个request中的参数值去指定 Bean 的指定的属性值
- ◆ property用于指定 Bean 的属性名
 - 即用request中的参数值去匹配那个属性
- ◆ Bean 属性和 request 参数的名字相同, 否则需要用 param 指定
 - 关于 param 参见下一个 property-expression 写法
- ◆ 如果request对象的参数值中是空值
 - 那么对应的Bean属性将不被设定任何值

The setProperty Action

■ About property-expression

- `property="propertyName" param="parameterName"`
 - ◆ `property` 属性同前面一样，用以指定 Bean 的属性名
 - ◆ `param` 用于指定 `request` 中的参数名
 - 当Bean中属性名与request对象中参数名不同时，必须用 `param` 指定 `request` 中的参数名
 - 如果二者同名，只需指定 `property` 即可(同上一种写法)
 - ◆ 如果 `request` 对象的参数值为空值，则对应的Bean属性将不被设置任何值

The setProperty Action

■ About property-expression

- `property="propertyName" value="propertyValue"`
 - ◆ 可以再运行时使用一个表达式来匹配 Bean 的属性
 - ◆ property 属性依然同前
 - ◆ value 是个可选属性，使用指定的值来设定 Bean 的指定属性
- 可以使用字符串、表达式来设定 Bean 的值
 - ◆ 字符串：调用相应包装类的 `valueOf` 方法自动转换为 Bean 属性的类型
 - ◆ 表达式：表达式的类型必须跟将要匹配的 Bean 的属性的类型一致
 - ◆ 如果参数值为空，那么对应的属性值也不会被设定

提示：不能在同一个 `<jsp:setProperty>` 中同时使用 `param` 和 `value`

The setProperty Action

■ <jsp:setProperty>的两种用法

■ 用法一

- 在 <jsp:useBean>之后使用 <jsp:setProperty>

```
<jsp:useBean id="bean" class= ... />
<jsp:setProperty name="bean" property= ... />
```

■ 用法二

- 在 <jsp:useBean> 标签内部使用 <jsp:setProperty>

```
<jsp:useBean id="bean" class= ... >
  <jsp:setProperty name="bean" property= ... />
<jsp:useBean>
```

Part 5 : Expression Language

1

What is EL

2

Operator . and []

3

Reserved Word

4

Implicit Objects

5

Operators in EL

What is EL

■ Expression Language , EL

- 表达式语言
- 原本是 JSTL 1.0 为方便存取数据所自定义的
 - ◆ 当时的 EL 只能在 JSTL 标签中使用(一般在属性中使用)
 - ◆ 比如: <c:out value=" \${ 7527 + 2474 } " >
- 从 JSP 2.0 开始, EL 被正式称为标准规范之一
 - ◆ 只要是支持 Servlet 2.4 / JSP 2.0 的容器, 都可以在 JSP 上直接使用
- 除了 JSP 2.0 建议使用 EL 外, JSF 也考虑将 EL 纳入规范
 - ◆ JSF : Java Server Faces

■ Syntax

`${ expression }`

- Example
 - ◆ `${ userList }`

Operator . and []

EL 提供 . 和 [] 运算符来存取数据

- 二者作用都是取出某个变量的值，不同之处在于使用方法和范围

. 运算符

- 使用时直接在某个变量之后跟 . 来访问该变量内部的属性或方法

- 比如

- User 类内部有 `getName()` 方法，其中返回 name 属性的值
 - 则可以使用 `${ user.name }` 可以获取 user 的 name 属性的值

- 规律：

- 如果 A 类中有 `getXyz()` 方法，设 `getXyz()` 方法有返回值
 - 若存在对象 a 是 A 类型，则可以使用 `${ a.xyz }` 获取 `getXyz()` 方法的返回值

[] 运算符

- 可以使用类似于 . 运算符的作用，形式为 `${ user["name"] }`
- 也类似于访问数组元素的风格使用，形式为 `${ list[1] }`

Reserved Word

EL中的保留字

- 类似于 Java 中的保留字
- 程序员不能够在 EL 中再度单独使用这些保留字

EL中的保留字有

and	eq	gt	true
or	ne	le	false
no	lt	ge	null
instanceof	empty	div	mod

Implicit Objects

EL中的内置对象

隐含对象	类型	说明
pageContext	javax.servlet.ServletContext	表示此JSP的PageContext
pageScope	java.util.Map	取得Page范围的属性名称所对应的值
requestScope	java.util.Map	取得Request范围的属性名称所对应的值
sessionScope	java.util.Map	取得Session范围的属性名称所对应的值
applicationScope	java.util.Map	取得Application范围的属性名称所对应的值
param	java.util.Map	如同ServletRequest.getParameter()

Implicit Objects

EL中的内置对象

隐含对象	类型	说明
paramValues	java.util.Map	如同 ServletRequest.getParameterValues(String name)。回传String[]类型的值
header	java.util.Map	如同ServletRequest.getHeader(String name)。回传String类型的值
headerValues	java.util.Map	如同 ServletRequest.getHeaders(Stringna me)。回传String[]类型的值
cookie	java.util.Map	如同HttpServletRequest.getCookies()
initParam	java.util.Map	如同 ServletContext.getInitParameter(Stri ng name)。回传String类型的值

Implicit Objects

EL中的内置对象的分类

与范围有关的内置对象

- ◆ applicationScope 、 sessionScope 、 requestScope 、 pageScope
- ◆ 变量的查找顺序为 p --> r --> s --> a

与输入有关的内置对象

- ◆ param 、 paramValues

其他内置对象

- ◆ cookie
- ◆ header
- ◆ headerValues
- ◆ initParam
- ◆ pageContext

Operators in EL

■ Arithmetic Operators

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /、div
- Remainder: % 、 mod

■ Relational Operators

- == 、 eq
- != 、 ne
- < 、 lt
- > 、 gt
- <= 、 le
- >= 、 ge

Operators in EL

Logic Operators

- && 、 and
- || 、 or
- ! 、 not

Empty Operators

- empty
 - ◆ 用在一个变量的前面
 - ◆ 作用是判断改变量是否为 null 或 为空(侧重于集合类元素中没有元素)

Operators in EL

Operator Precedence (运算符优先级)

- [] .
- ()
- -(unary) 、 not 、 ! 、 empty
- * 、 / 、 div 、 % 、 mod
- + 、 - (binary)
- < 、 > 、 <= 、 >= 、 lt 、 gt 、 le 、 ge
- == 、 != 、 eq 、 ne
- && 、 and
- || 、 or

Part 6 : Custom Tags



Custom Tag Overview



Defining Custom Tags



Advanced Features of Custom Tag



Custom Function

Custom Tag Overview

■ What is a Custom Tag

- 用户自定义的 Java 语言元素
 - ◆ 属于 Java 范畴, 不属于 HTML 体系
- 封装周期性任务
 - ◆ 定义标签大都是为了重复使用
- 当包含自定义标签的 JSP 页面转换为 servlet 时
 - ◆ 这个标签就转换为一个名为 tag handler 的对象上的操作
 - ◆ 之后当 JSP 页面的 servlet 执行时, Web 容器就调用这些操作
- 可用作
 - ◆ 操作内置对象
 - ◆ 处理表单数据
 - ◆ 访问数据库以及其他企业级服务 等
- 分布在"自定义"标签库中
 - ◆ 一般被定义在叫做 tld 的文件中
 - ◆ 这个 tld 文件本质上是个 xml 文件

Custom Tag Overview

Custom Tag Features

- 通过从调用页面传递的属性进行定制
- 访问JSP页面可以使用的所有对象
- 修改由调用页面生成的响应
- 彼此通信
 - ◆ 可以创建并初始化JavaBean组件
 - ◆ 在一个标签中创建引用该bean的变量
 - ◆ 再在另一个标签中使用这个bean
- 彼此嵌套，可以在JSP页面中实现复杂的交互交互

Custom Tag Overview

■ Why Custom Tag

- 在 HTML 页面中插入 JSP 脚本的不足之处
 - ◆ JSP 脚本非常丑陋, 难以阅读
 - ◆ JSP 脚本和 HTML 代码混杂, 维护成本高
 - ◆ HTML 页面中嵌入 JSP 脚本, 导致美工人员难以参与开发
- 使用自定义标签
 - ◆ 大大简化 Web 应用的表现层开发
 - 非常优秀的表现层组件技术
 - 绝大多数的 MVC 框架的视图技术都是采用自定义标签, 如Struts
 - ◆ 在JSP页面上实现单独的业务逻辑或其他功能
 - ◆ 创建业务逻辑的开发者创建定义标签
 - ◆ 封装业务逻辑
 - ◆ 可以反复重用、易于维护
 - ◆ 逻辑与页面相对分离, 便于分工

Custom Tag Overview

■ Types Of Tag (Two types)

■ Has body

- ◆ 有标签体，必须位于开始标记和结束标记之间
- ◆ 内容包括
 - 自定义标记或核心标记
 - HTML 文本
 - 标签依赖的主体内容
- ◆ 比如
 - <html:submit> OK </html:submit>

■ Has no body

- ◆ 没有标签体
- ◆ 比如 <html:errors />

Custom Tag Overview

■ Tag with Attributes

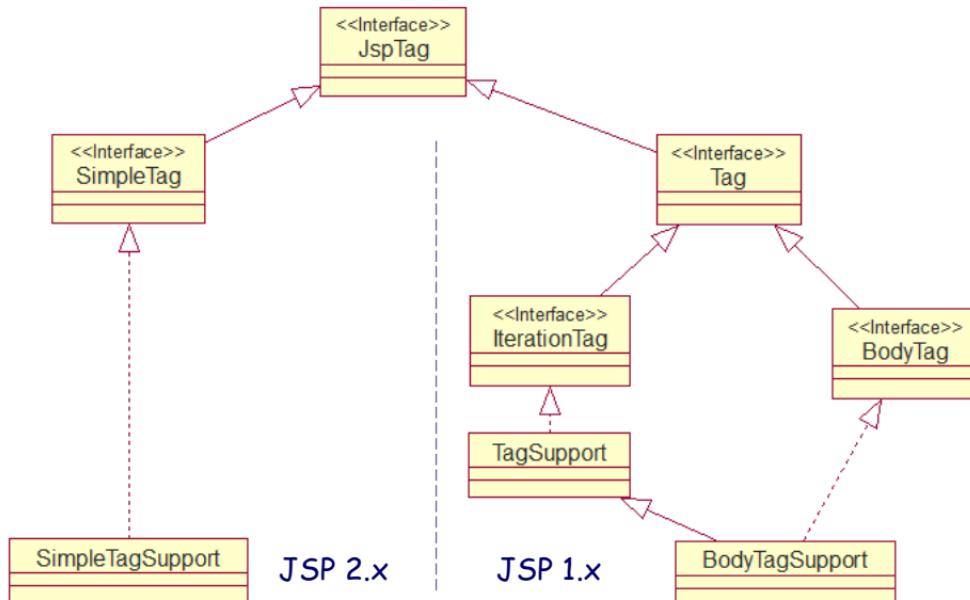
- Attributes are listed in the start tag
 - ◆ 属性列被定义在自定义标记的开始标记中
- Attribute has the syntax attr="value"
 - ◆ 定义属性的语法为 attributeName="attributeValue"
- Attribute values serve to customize the behavior of a custom tag
 - ◆ 属性值可以定制一个自定义标记的行为
- The types of a tag's attribute are specified in a TLD file
 - ◆ 在 TLD 文件中可以指定标记的属性的类型

Defining Custom Tags

- Three Things Make up Custom Tag Architecture
 - Tag handler class
 - ◆ 标记处理器类
 - ◆ 定义标签的行为
 - Tag Library Descriptor (TLD)
 - ◆ 标签库描述符(tld)
 - 建立一个 *.tld 文件
 - » 每个 *.tld 文件对应一个标签库
 - » 每个标签库对应多个标签
 - ◆ 通过XML元素标记处理程序类
 - ◆ tld 文件必须放在 WEB-INF 目录或者其任意子目录
 - JSP file
 - ◆ 在 JSP 文件中使用自定义标签

Defining Custom Tags

- Step 1 : Writing Tag Handler Class
 - Tag Class Hierarchy



Defining Custom Tags

Step 1 : Writing Tag Handler Class

- Extend SimpleTagSupport
 - ◆ 如果自定义标签包含属性
 - ◆ 那么对应的标签类中也要有相应的属性，名称还要完全相同
 - ◆ 每个属性都必须有对应的 getter 和 setter 方法
- 重写 doTag() 方法
 - ◆ 这个方法负责生成页面内容

```
// 标签处理类，继承 SimpleTagSupport 父类
public class TagClassName extends SimpleTagSupport {
    // 重写 doTag 方法，该方法在标签结束生成页面内容
    public void doTag() throws JspException, IOException {
        // 获取页面输出流，并输出字符串
        this.getJspContext().getOut().write(".....");
    }
}
```

Defining Custom Tags

Step 1 : Writing Tag Handler Class

Example

```
package com.malajava.customtag;

import java.io.IOException;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class HelloWorldTag extends SimpleTagSupport {

    @Override
    public void doTag() throws JspException, IOException {
        JspWriter out = this.getJspContext().getOut();
        out.write( "Hello, this is my first tag!" );
    }
}
```

Defining Custom Tags

Step 2 : Writing TLD File

Tag Library Descriptor

- 使用 XML 语言描述，包括
 - 标记名称
 - 标记主体
 - 标记的属性
 - 标记对应的处理程序类
- 容器通过这个文件找到标记对应的程序处理类
- 通常被放置在 WEB-INF 目录下
- 在 JSP 中使用时，通过 taglib 指令的 uri 属性指定

What is Tag Library

- 一般是一组相关的标签组成
 - 标签通常被封装在标签库中 (对应一个 tld 文件，一般被打成 jar 包)
- 一个基本的标签库，如 malajava.tld

Defining Custom Tags

Step 2 : Writing TLD File

Example for TLD

```
<?xml version="1.0" encoding="UTF-8"?>

<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 web-jsptaglibrary_2_0.xsd"
    version="2.0">

    <tlib-version>1.0</tlib-version>
    <short-name>malataglib</short-name>
    <uri>http://www.malajava.com/malataglib</uri> <!-- 定义该标签库的URI -->
    <!-- 定义第一个标签 -->
    <tag>
        <name>hello</name><!-- 定义标签名 -->
        <!-- 定义标签处理类 -->
        <tag-class>com.malajava.customtag.test.HelloWorldTag</tag-class>
        <!-- 定义标签体为空 -->
        <body-content>empty</body-content>
    </tag>
</taglib>
```

Defining Custom Tags

Step 2 : Writing TLD File

■ <tag>各子元素的意义与作用

- ◆ <description> 可选, 标签的描述信息
- ◆ <name> 必需, 定义JSP页面引用的标签名称(标签后缀)
- ◆ <tag-class> 必需, 标识实现标签处理类的完全限定名(含包名的类名)
- ◆ <body-content> 必需, 告诉容器如何处理标签开始和结束之间的内容
 - <body-content> 取值可以是 empty、scriptless、tagdependent、JSP
 - empty : 不允许任何内容出现在标签体内(用于定义没有标签体的标签)
 - scriptless : 允许标签中有 JSP 内容, 但不能包含 <% ... %> 和 <%= ... %> 之类
 - tagdependent : 允许此标签像其主体一样有任何类型的内容
 - » 但是这些内容根本不会被处理, 并会完全忽略
 - » 由标签处理类开发人员来决定访问这些内容, 并对其进行处理
 - » 比如其中包含一个允许 JSP 开发人员执行的 SQL 语句
 - JSP : 允许在标签体中放置 JSP 内容

Defining Custom Tags

Step 3 : Use Custom Tags

导入标签库

```
<%@ taglib prefix="tagPrefix" uri="tlduri" %>
```

- uri 用于指定 tld 中 <uri> 中的字符串值
- prefix 用于指定标签的前缀，这个名称任意取

使用标签

```
<tagPrefix:tagName />
```

- 其中 tagPrefix 要与 taglib 中的 prefix 的值写一致
- 而 tagName 就是 tld 文件中 <tag> 内部 <name> 标签内部的字符串值

Defining Custom Tags

- Step 3 : Use Custom Tags
- Example for JSP

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%@ taglib prefix="malajava" uri="http://www.malajava.com/malataglib" %>

<!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>index</title>
  </head>
  <body>

    <malajava:hello/>

  </body>
</html>
```

Defining Custom Tags

Tag with Attribute

Step 1 : Writing tag handler class

```
package com.malajava.customtag;

import java.io.IOException;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class TagWithAttribute extends SimpleTagSupport {
    private int num1; // 将要在自定义标签中出现的属性名称
    private int num2; // 将要在自定义标签中出现的属性名称

    @Override
    public void doTag() throws JspException, IOException {
        this.getJspContext().getOut().write(num1 + " * " + num2 + " = " + (num1 * num2));
    }
    // 此处没有列出 num1 和 num2 的 getter 和 setter , 请自行补全(必须)
}
```

Defining Custom Tags

■ Tag with Attribute

■ Step 2 : Writing TLD file

```
.....  
<tag>  
  <name>counter</name> <!-- 定义标签名 -->  
  <!-- 定义标签处理类 -->  
  <tag-class>com.malajava.customtag.TagWithAttribute</tag-class>  
  <body-content>empty</body-content> <!-- 定义标签体为空 -->  
  <attribute>  
    <name>num1</name> <!-- num1 属性的名字 -->  
    <required>true</required>  
    <rexprvalue>true</rexprvalue>  
  </attribute>  
  <attribute>  
    <name>num2</name> <!-- num2 属性的名字 -->  
    <required>true</required>  
    <rexprvalue>true</rexprvalue>  
  </attribute>  
</tag>  
.....
```

Defining Custom Tags

Tag with Attribute

- <attribute> 标签中各个子元素的意义及作用
 - ◆ <name> 必需, 定义属性名称
 - 严格区分大小写, 与 Tag Handler Class 里的 getter 和 setter 对应
 - ◆ <required> 可选, 指定是必须始终提供该属性
 - false 是默认值, 表示并不是一定要提供该属性
 - true 是可选值之一, 表示必须始终提供该属性
 - ◆ <rtexprvalue> 可选, 指出属性值是一个动态确定的值还是一个固定的值
 - true 表示属性的值是个变量, 可以是 <%=expression %> 或 \${bean.value}
 - 注意, 标签体包含 JSP 表达式是不合法的
 - 但是标签的属性值却可以通过表达式或 EL 来动态确定
 - false 表示属性的值是个固定的字符串值, 这是默认值
 - 因为该元素默认值是 false, 通常忽略该元素

Defining Custom Tags

Tag with Attribute

Step 3 : Use custom tag

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%@ taglib prefix="malajava" uri="http://www.malajava.com/malataglib" %>

<!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>Insert title here</title>
  </head>
  <body>
    <malajava:counter num2="23" num1="32"/>
  </body>
</html>
```

Defining Custom Tags

Tag with Body

Step 1 : Writing tag handler class

```
package com.malajava.customtag;

import java.io.IOException;
import java.io.StringWriter;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class TagWithSingleBody extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {
        //获得当前标签的标签体(JspFragment对象)
        JspFragment jspFragment = this.getJspBody();
        //输出标签体内的内容( invoke 需要的参数是 Writer 类型, 如果是 null 默认采用内置对象 out)
        jspFragment.invoke( null );
    }
}
```

Defining Custom Tags

Tag with Body

Step 2 : Writing TLD file

```
.....  
<tag>  
  <name>tagWithSingleBody</name>  
  <tag-class>com.malajava.customtag.TagWithSingleBody</tag-class>  
  <body-content>tagdependent</body-content>  
</tag>  
.....
```

Step 3 : Use custom tag

```
<malajava:tagWithSingleBody>Welcome to malajava.com !</malajava:tagWithSingleBody>
```

Defining Custom Tags

■ Create Tag File

- JSP 2.0 以后允许定义基于 JSP 的标签文件
 - ◆ 该文件内部是一个 JSP 片段
 - ◆ 它有一些特殊的指令和一个以 .tag 为结束的文件名
 - ◆ 必须存放在 WEB-INF/tags 目录之下
- 使用
 - ◆ Step 1 : Writing Tag file
 - ◆ 文件名就是将来的标签名, 比如创建 hello.tag 则使用时 hello 就是标签名

```
<%= "hello" %>
```

- ◆ Step 2 : Use tag file

导入需要的库

```
<%@ taglib prefix="malajava" tagdir="/WEB-INF/tags" %>
```

使用标签

```
<malajava:hello />
```

Defining Custom Tags

■ Create Tag File

■ 定义属性

- ◆ 注意，标签中传递来的属性值都是字符串类型，需要处理

```
<%@ attribute name="attributeName" required="false" %>  
<%= attributeName %>
```

■ 使用标签体

```
.....  
<jsp:doBody />  
.....
```

Advanced Features of JSP Custom Tag

■ Dynamic Values for Attribute

- 自定义标签允许使用运行时变量来确定属性的值，比如

```
<malajava:table row="\$\{ row \}" cell="<%= cells %>" />
```

- 必须在 tld 文件中设置可以接受运行时变量的值

```
.....  
<tag>  
  <attribute>  
    <name>num1</name> <!-- num1 属性的名字 -->  
    <required>true</required>  
    <rexprvalue>true</rexprvalue>  
  </attribute>  
</tag>  
.....
```

Advanced Features of JSP Custom Tag

Custom Function

EL Function

- EL 函数是 JSP 2.0 规范新增加的特性
- 它允许在 JSP 页面上，使用 EL 来调用函数

```
 ${ prefix:functionName( arguments ) }
```

Custom Function

Custom Function

Step 1 : Create Java Class

- 创建一个 Java 类，其中必须包含一个 public static 的方法

```
public class ClassName {  
    public static void methodName(){  
        .....  
    }  
}
```

Step 2 : Mapping

- 在 TLD 中声明这个方法(建立映射)
- function-signature 中返回类型以及参数类型需要是全限定名

```
<function>  
    <name>show</name>  
    <function-class>com.malajava.customtag.Function</function-class>  
    <function-signature>void show()</function-signature>  
</function>
```

Custom Function

Custom Function

- Step 3 : Using Function
 - 在 JSP 页面中使用

```
.....  
${malajava:show() }  
.....
```

Part 7 : Java Standard Tag Lib

1

Introduction to JSTL

2

Core Tags

3

Function Tags

4

Fmt Tags

5

SQL Tags

What is JSTL

■ JSTL 是一个标准的标签库

- 它由 Sun 公司制定，不需要其它第三方支持
- 它封装了 JSP 应用程序中的核心功能
 - ◆ 迭代操作、条件判断、操作 XML、访问数据库、国际化和数据格式化等
- 将来的版本中可能添加更多的功能

■ 为什么要使用 JSTL

- 不需要再重新编写，直接使用即可
- 学习和掌握标准的标签库是学习 Java 平台开发的趋势
 - ◆ 绝大多数框架都使用了大量的标签库
- 使应用程序的可移植性增强

JSTL Tag Libraries

- Core (prefix: c)
 - 访问变量, 流程控制, URL 管理
- XML (prefix: x)
 - XML 核心, 流程控制, 转换
- Internationalization (I18N) (prefix: fmt)
 - 国际化, 信息格式化, 数字格式化, 日期格式化
- Database (prefix: sql)
 - 执行 SQL 查询和更新
- Functions (prefix: fn)
 - 集合操作、字符串操作

JSTL Tag Library Jar Files

- JSTL 被分布在两个 jar 文件中
 - standard.jar
 - jstl.jar
- 使用这些 jar 包
 - 容器加载，并让所有应用程序使用这些包
 - ◆ 把这两个包复制到 TOMCAT_HOME/lib 目录下
 - ◆ 这种方式，将对所有的应用起作用
 - 只针对当前应用使用
 - ◆ 复制到 yourApp/WEB-INF/lib 中
 - ◆ 这种方式，仅仅对本应用起作用

Core Tags

Core Tags Types

- 变量相关
 - ◆ <c:set />、<c:remove />
- 条件语句
 - ◆ <c:if />、<c:choose />、<c:when />、<c:otherwise />
- 遍历
 - ◆ <c:forEach />、<c:forTokens />
- URL 相关
 - ◆ <c:url />、<c:import />、<c:redirect />
- 其它
 - ◆ <c:out />、<c:catch />

Core Tags URI

- <http://java.sun.com/jsp/jstl/core>

■ <c:set>

■ Intention

- ◆ 可以声明一个不存在的对象
- ◆ 也可以修改一个已经存在的对象
- ◆ 还可以修改某个对象的某个属性
 - 前提是该属性是可写的(有 setter 方法)

■ Syntax

```
<c:set var="test" value="by value property" ></c:set>
```

Or

```
<c:set var="test" > by body </c:set>
```

■ <c:set>

■ Attributes

- ◆ var : 需要设置的对象的名称, 若不存在则生成之, 有则改之
 - var 只能设置 Double、Integer、Float、String 等类型数据
 - » 创建一个变量并赋予值 <c:set var="name" value="华安"></c:set>
 - » 修改已经存在的name变量的值 <c:set var="name" value="九五二七"></c:set>
 - 但不能操作其他 JavaBean 或 Map 等复杂类型的对象的内部数据
 - » 创建一个变量, 并把 user 赋给该变量 <c:set var="v" value="<%=" user %>">
 - » 不可以使用var修改这个变量的内部数据 var="v" property="id" value="9527"
- ◆ value : 就是要给 var 指定的对象设置的值
 - 可以是 EL 表达式, 也可以是具体的一个值
 - 有时候也可以是 JSP 表达式 <%= %>
- ◆ scope : 声明该对象存在的范围, 默认是 page
 - 可选范围有: page、request、session、application

■ <c:set>

■ Attributes

- ◆ target : 作用等同于 var , 不同的是操作的对象
 - target 专门用于操作 JavaBean 和 Map 等复杂类型的数据
 - target 跟 var 不能同时出现在 c:set 标签中
 - target 用以引用一个已经存在的对象
 - » 严格来说, 只用于修改已经存在的 Java Bean 或者 Map , 而不能创建它们
 - target 属性的值必须是 EL 表达式或者 JSP 表达式
 - » 当某个已经存在的对象存在于P、R、S、A的某个范围或者使用 jsp:useBean 创建, 使用EL
 - » 当某个已经存在的对象是在Java代码块中声明和创建时, 一般使用 <%= %>
- ◆ property :
 - 需要跟 target 连用, 作用是指定 JavaBean 的属性的名称
 - 或者指定 Map 中 key 的名称

Core Tags

■ <c:set>

■ Example

```
<%@page language="java"    contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core"    prefix="c"%>
<html>
  <head>
    <meta    http-equiv="Content-Type"    content="text/html; charset=UTF-8">
    <title>统计访问次数</title>
  </head>
  <body>
    <c:set var="totalCount" value="${totalCount +1 }" scope="application"></c:set>
    <c:set var="count" value="${ count + 1 }" scope="session"></c:set>
    本站总访问人数: ${totalCount } <br/>
    其中您访问次数: ${count }
  </body>
</html>
```

Core Tags

■ <c:remove>

■ Intention

- ◆ 删除指定 范围中的指定变量

■ Syntax

```
<c:remove var="test" scope="session" />
```

■ Attributes

- ◆ 只有 var 和 scope 两个属性，作用跟 c:set 完全一样

■ Example

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

```
<c:remove var="totalCount" scope="application" />
```

```
<c:remove var="count" scope="session" />
```

Core Tags

■ <c:if>

■ Intention

- ◆ 用于条件判断，类似于 Java 中的 if 语句(但c:if没有 else 选项)

■ Syntax

```
<c:if test="条件判断" />
```

■ Attributes

- ◆ 只有 test 一个属性，用于返回一个 boolean 值

■ Example

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

```
<c:if test="${param.action == 'add'}">添加操作</c:if>
```

```
<c:if test="${param.action == 'edit'}">修改操作</c:if>
```

Core Tags

■ <c:choose>

■ Intention

- ◆ 用于条件选择，必须跟 <c:when> 和 <c:otherwise> 连用
- ◆ <c:when> 类似于 Java 中的 if 语句
- ◆ <c:otherwise> 类似于 Java 中的 else 语句

■ Syntax

```
<c:choose>
  <c:when test="条件判断">
    满足条件执行的语句
  </c:when>
  <c:otherwise>
    不满足条件执行的语句
  </c:otherwise>
</c:choose>
```

Core Tags

■ <c:choose>

■ Example

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

<c:if test="${!empty param.name}">Hello ${param.name} </c:if>
<c:if test="${!empty param.age}">
<c:choose>
  <c:when test="${param.age < 18}">你还是个孩子. </c:when>
  <c:when test="${param.age >=18 && param.age < 37}">你是个年青人. </c:when>
  <c:otherwise>你是位老人. </c:otherwise>
</c:choose>
</c:if>
```

■ <c:forEach>

■ Intention

- ◆ 用于遍历集合、数组，或者执行循环操作

■ Attributes

- ◆ items : 需要被迭代的集合元素
 - 当某个集合可以使用 EL 获得时，通过 EL 指定
 - 当某个集合不可以使用 EL 获得时，可以考虑使用 <%= %> 指定
- ◆ var : 定义变量的名称
- ◆ varStatus : 用于记录当前被遍历的对象的信息
- ◆ begin : 定义开始位置
- ◆ end : 定义结束位置
- ◆ step : 定义步长

■ 可以迭代的集合的类型

- ◆ java.util.Collection
- ◆ java.util.Map

■ <c:forEach>

■ varStatus 属性的属性

- ◆ index : 返回当前元素是第几个, 从 0 开始计数
- ◆ count : 返回已经遍历了多少个元素, 从1开始计数
- ◆ first : 返回当前对象是否是第一个对象
- ◆ last : 返回当前对象是否是最后一个对象
- ◆ current : 返回当前被遍历的对象
- ◆ begin : 返回 forEach 标签中 begin 属性的值
- ◆ end : 返回 forEach 标签中 end 属性的值
- ◆ step : 返回 ForEach 标签中 step 属性的值

Core Tags

■ <c:forEach>

■ Example

- ◆ 用做循环

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:forEach var="n" begin="1" end="10" step="2">
    ${n}
</c:forEach>
```

- ◆ 遍历 List

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:forEach var="user" items="${userList}">
    ${user.id} - ${user.name} - ${user.age} <br />
</c:forEach>
```

Core Tags

■ <c:forEach>

■ Example

- ◆ 遍历 Map

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:forEach var="item" items="${ header }" >
    ${ item.key } - ${ item.value } <br />
</c:forEach>
```

- ◆ 使用 varStatus 属性

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:forEach var="user" items="${ userList }" varStatus="heihei">
    <h4 style="color : ${ heihei.index % 2 == 1 ? 'blue' : 'red' }">
        ${ heihei.current.id } - ${ heihei.current.name }
    </h4>
</c:forEach>
```

Core Tags

■ <c:forTokens>

■ Intention

- ◆ 使用方法以及属性跟 c:forEach 几乎完全一样
- ◆ 区别于 c:forEach 的是 items 只能是字符串

■ Example

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:forTokens var="v" varStatus="h" begin="2" end="5" step="2" delims="," items="zhangsan , lisi ,wangwu , zhaoliu ,shaseng ,bajie ,wukong,tangseng" >
    ${ h.index } - ${ v } <br />
</c:forTokens>
```

Core Tags

■ <c:out>

■ Intention

- ◆ 输出一个值到页面上

■ Syntax

```
<c:out value="value"  
       [escapeXml="{true|false}"] [default="defaultValue"] />
```

■ Example

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  
<c:out value="${param.action}" />
```

Core Tags

■ <c:import>

■ Intention

- ◆ 导入网络资源，类似于 <jsp:include >

■ Attributes

- ◆ url : 需要包含的 url
- ◆ charEncoding : 指定目标 url 采用的编码
- ◆ var : 存储 url 对应的内容的变量
- ◆ varReader : 存储 url 对应内容的 java.io.Reader 变量
 - varReader 不能与 var 或 scope 同时存在
- ◆ context : 当访问本应用内的网页时使用
 - context 用于指定访问的文件的路径，那么 url 必须以 / 开头，指定文件名
- ◆ scope : 指定内容存放的范围

Core Tags

■ <c:url>

■ Intention

- ◆ 相当于 `response.encodeURL()` , 用于URL 重写

■ Syntax

```
<c:url value="/index.jsp" />
```

■ <c:redirect>

■ Intention

- ◆ 实现 Redirect 重定向功能

■ Syntax

```
<c:redirect url="http://www.malajava.com/" ></c:redirect>
```

Core Tags

■ <c:param>

- Intention
 - ◆ 用于传递参数
- Syntax

```
<c:param name="paramName" value="paramValue" />
```

■ Example

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:redirect url="http://www.mala.java.com/index.jsp" >
    <c:param name="username" value="wukong" ></c:param>
</c:redirect>
```

Core Tags

■ <c:catch>

■ Intention

- ◆ 用于捕获异常，只有一个 var 属性

■ Syntax

```
<c:catch var="e"> 需要捕获异常的代码 </c:catch>
```

■ Example

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<c:catch var="e" >
    <c:set target="someBean" property="someProperty"
           value="somevalue"></c:set>
</c:catch>
<c:if test="${ e != null }">
    程序抛出了异常 ${ e.class.name }, 原因是 ${ e.message }
</c:if>
```

Function Tags

■ Intention

- 相当于Java中的方法，用在 EL 表达式中
- 因此有人也称之为是 EL 函数

■ URI

- `http://java.sun.com/jsp/jstl/functions`

■ Syntax

```
 ${ fn:length( "abc123def567hjk89" ) }
```

■ Example

```
<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
${ fn:contains( 'abc123def456hjk789' , 'jk' ) }
```

Function Tags

- fn:length()
 - 获取字符串、数组、集合的长度
- fn:toUpperCase(String str)
 - 把字符串 str 中的英文转成大写
- fn:toLowerCase(String str)
 - 把字符串 str 中的英文转成小写
- fn:substring(String str , int startIndex , int endIndex)
 - 从 str 中截取子串
- fn:substringBefore(String str , String a)
 - 截取子串，从头开始截取，到指定子串 a 结束
- fn:substringAfter(String str , String a)
 - 截取子串，从指定子串 a 开始到结束

Function Tags

- fn:trim(String str)
 - 截取字符串 str 的首尾空白
- fn:replace(String str , String a , String b)
 - 使用指定的字符串 b 替换 str 中的 a 字符串
- fn:indexOf(String str , String a)
 - 从 str 中查找 a 第一次出现的位置, 如果不存在, 返回 -1
- fn:startsWith(String str , String a)
 - 判断 str 是否以 a 开头
- fn:endsWith(String str , String a)
 - 判断 str 是否以 a 结束

Function Tags

- fn:contains(String a , String b)
 - 判断字符串 a 中是否包含子串 b (区分大小写)
- fn:containsIgnoreCase(String a , String b)
 - 判断字符串 a 中是否包含子串 b (忽略大小写)
- fn:split(String source , String seperator)
 - 把字符串 source , 根据指定的 seperator 进行拆分
- fn:join(String[] array , String seperator)
 - 把数组中的元素连接成字符串, 新字符串中用 seperator 分隔
- fn:escapeXml(String source)
 - 忽略 source 中的 XML 字符

Internationalization & Text Formatting Tags

■ Intention

- 实现国际化
- 对数字或日期进行格式化操作

■ URI

- <http://java.sun.com/jsp/jstl/fmt>

■ Syntax

```
<fmt:requestEncoding value="UTF-8" />
```

Internationalization & Text Formatting Tags

■ <fmt:requestEncoding> 标签

- 用于设置 编码
- 相当于 `request.setCharacterEncoding()` 操作
- 该标签对于 POST 请求可以起作用
- 对于 GET 请求
 - ◆ 需要修改 `TOMCAT_HOME/conf/server.xml`
 - ◆ 在没有被注释掉的 `Connector` 标签中(8080)
 - 添加 `URIEncoding="UTF-8"` 和 `useBodyEncodingForURI="true"`
 - ◆ 否则 `requestEncoding` 标签不起作用

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<fmt:requestEncoding value="UTF-8"/>
<form action="${pageContext.request.requestURI}" method="post">
    关键字: <input type="text" name="key" />
    <c:out value="${param.key}" default="请输入关键字" /> <input type="submit" />
</form>
```

Internationalization & Text Formatting Tags

■ <fmt:formatDate> 标签

- 用于将日期转化为指定格式的字符串
- 相当于 `java.text.DataFormat`

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<jsp:useBean id="currentDate" class="java.util.Date"/>
<fmt:formatDate value="${currentDate}" /><br>
<fmt:formatDate value="${currentDate}"
    pattern="yyyy/MM/dd HH:mm:ss:sss"/>
```

Internationalization & Text Formatting Tags

■ <fmt:formatDate> 标签

■ formatDate 中的属性

- ◆ value 需要格式化的日期
- ◆ type 指定该日期的类型是日期还是时间
 - 取值可以是 date、time、both
- ◆ dateStyle 日期的格式
 - 取值可以是 short、medium、long、full
- ◆ timeStyle 时间的格式
 - 取值可以是 short、medium、long、full
- ◆ timeZone 时间所在的时区
- ◆ pattern 指定日期的格式
- ◆ var 将格式化后的日期输出到 var 变量中
- ◆ scope 声明 var 变量的作用域
 - 取值可以是 page、request、session、application

Internationalization & Text Formatting Tags

■ <fmt:parseDate> 标签

- 将字符串转化为时间
- 相当于 `java.text.DateFormat` 的 `parse` 方法
- `parseDate` 拥有 `formatDate` 标签的所有的属性
 - ◆ 这些属性的使用方法跟 `formatDate` 完全一样
- 另外还有一个属性 `parseLocale`
 - ◆ 用于指明按照什么 `Locale` 来处理
- 举例

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<fmt:parseDate value="2011-08-05" parseLocale="zh"></fmt:parseDate>
<fmt:parseDate value="2011-08-05 22:58:33"
                parseLocale="zh" timeZone="America/Los_Angeles"
                pattern="yyyy-MM-dd HH:mm:ss"></fmt:parseDate>
```

Internationalization & Text Formatting Tags

■ <fmt:formatNumber> 标签

- 把数字格式化为指定的格式字符串
- 相当于 `java.text.NumberFormat` 的 `format` 方法
- 举例

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

<fmt:formatNumber value="8888" type="currency"/>

<fmt:formatNumber value="8888" type="percent"/>
```

Internationalization & Text Formatting Tags

■ <fmt:formatNumber> 标签

■ formatName 中的属性

- ◆ value 需要被格式化的数字
- ◆ type 声明数字的类型，可以是
 - 数字 number
 - 货币 currency
 - 百分数 percentage (Apache JSTL 中是 percent)
- ◆ pattern 设置数字格式
 - 格式比如 0000.00，可以参考 java.text.NumberFormat 类
- ◆ currencyCode 指定 ISO 4217 规定的货币代码
 - 美元为 USD、人民币为 CNY
- ◆ currencySymbol 输出货币时的货币符号
 - 美元默认为 \$，人民币默认为 \$
 - 仅当 type = currency 时有效

Internationalization & Text Formatting Tags

■ <fmt:formatNumber> 标签

■ formatName 中的属性

- ◆ groupingUsed 是否输出分隔符
 - 默认为 true , 就是输出分隔符
 - 如果设置为 false , 可以控制不输出分隔符
- ◆ maxIntegerDigits 设置数字的整数位的最大位数
 - 如果数字超过了最大位数限制, 那么就截掉高位
- ◆ minIntegerDigits 设置数字的整数位的最小位数
 - 如果不够则在高位补 0
- ◆ maxFractionDigits 设置数字的小数位数的最大位数
 - 如果超过了这个位数, 那么就四舍五入
 - 仅当 type = currency 时有效
- ◆ minFractionDigits 设置数字的小数位的最小位数
 - 如果不够, 那么低位补 0
- ◆ var 将格式化后的数字存储到 var 变量中
- ◆ scope 声明 var 变量的作用域

Internationalization & Text Formatting Tags

■ <fmt:parseNumber> 标签

- 把字符串 转换为 数字
- 相当于 `java.text.NumberFormat` 的 `parse` 方法
- 举例

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

<fmt:parseNumber value="1,000.00" />

<fmt:parseNumber value="1,000.00" parseLocale="de" />
```

Internationalization & Text Formatting Tags

■ <fmt:parseNumber> 标签

■ parseNumber 标签中的属性

- ◆ value 需要解析的字符串
- ◆ parseLocale 按照哪种 Locale 进行解析
- ◆ type 数据类型, 同 formatNumber 标签
- ◆ pattern 数据格式, 同 formatNumber 标签
- ◆ var 输出数值到 var 变量
- ◆ scope 声明 var 变量的作用域

Internationalization & Text Formatting Tags

■ <fmt: setLocale>

- 用于设定 Locale (Locale 是本地化的意思)
- Example
 - ◆ 获取所有国家的本地化信息

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page import="java.util.*" %>
<% request.setAttribute("localeList", Locale.getAvailableLocales()); %>
<jsp:useBean id="date" class="java.util.Date" %></jsp:useBean>
<table border="1">
  <tr>
    <th>Locale</th>          <th>Language </th>
    <th>Date and Time </th>    <th>Number </th>
    <th>currency </th>
  </tr>
```

Internationalization & Text Formatting Tags

■ <fmt: setLocale>

■ Example

- ◆ 获取所有国家的本地化信息(接上页)

```
<c:forEach var="locale" items="${localeList}">
    <fmt:setLocale value="${ locale }" />
    <tr>
        <td>${ locale.displayName }</td>
        <td>${ locale.displayLanguage }</td>
        <td><fmt:formatDate value="${ date }" type="both"/></td>
        <td><fmt:formatNumber value="8888.88"/></td>
        <td><fmt:formatNumber value="8888.88" type="currency"/></td>
    </tr>
</c:forEach>
</table>
```

■ Intention

- 封装了数据库访问的通用逻辑
- 允许在JSP上完成查询、更新、事务处理、设置数据源等操作

■ URI

- `http://java.sun.com/jsp/jstl/sql`

■ Syntax

```
<sql:query ... ... />
```

■ Tags

- `setDataSource`
- `query`、`param`
- `update`、`transaction`

■ <sql:setDataSource>

■ Intention

- ◆ 用于设置数据源，同时可以设置数据源的范围 (PRSA)

■ Attributes

◆ DataSource

- 如果用 String 表示 JNDI 名称空间中的 DataSource 名称
- 如果是 DataSource 则表示一个可以获取到的 DataSource 对象

◆ url : 所要访问的数据库的 URL

◆ driver : 数据库驱动类

◆ user : 数据库用户名

◆ password : 设置访问数据库的密码

◆ var : 用来标识这个 DataSource , 即名称

◆ scope : 设置这个 DataSource 的范围, 比如 session 等

■ <sql:query>

■ Intention

- ◆ 用来查询数据库

■ Attributes

- ◆ dataSource 作用与 setDataSource 中相同
- ◆ sql : 用于设置 SQL 语句
- ◆ maxRows : 设置查询结果的最大行数
- ◆ startRow : 设置查询结果开始的索引
- ◆ var : 用来标识存储查询结果的变量
- ◆ scope : 设置作用范围, 比如 request 、 session 等

 <sql:param>

■ Intention

- ◆ 设置 SQL 预处理语句中的参数
- ◆ 一般被套用在 query 或 update 标签的内部

■ Attributes

- ◆ value : 用来设定参数的值
- ◆ 也可以不使用 value 属性, 而在标签体中书写参数值

■ <set:update>

■ Intention

- ◆ 用来完成对数据库的更新操作

■ Attributes

- ◆ DataSource 作用与 setDataSource 相同
- ◆ sql : 设定需要执行的 SQL 语句
- ◆ var : 用来标识这个更新, 即一个名称
- ◆ scope : 设置这个 var 对应的变量的有效范围 (PRSA)

■ <sql:transaction>

■ Intention

- ◆ 为 query 和 update 建立事务处理的上下文

■ Attributes

- ◆ DataSource : 作用与 setDataSource 中相同
 - 使用了 transaction 标签, 内部的 update、query 不再需要 dataSource 属性
- ◆ isolation : 用来指定事务的隔离级别

Java Server Pages

---- cnhanxj@gmail.com