

第4章 JSP的语法和语义

本章将详细介绍JSP的语法和语义（JSP1.1）。如果读者接触过ASP或PHP，将会发现JSP的语法稍显复杂；另一方面，如果读者有Java语言程序设计的经验，就会觉得JSP相当简单，其实，作为J2EE的成员，JSP本来就已经成为Java的一部分了。

在JSP中，主要包含以下内容：

指令。指令提供该页的全局信息，例如，重要的状态，错误处理，是否是session的一部分等。

声明。页面范围的变量和方法的声明。

脚本。嵌入页面内java代码。

表达式。把表达式变成string型以便于包含在页面的输出中。

下面将一一介绍。

4.1 通用的语法规则

JSP的页面是由许多的“元素”组成的，本节描述的语法规则对于这些“元素”都是成立的，所以称之为“通用”的语法规则，也就是这些元素共有的特性。

4.1.1 元素的语法规则

大部分的JSP元素都基于“XML”的语法，这些基于“XML”语法的JSP元素一般来说都有一个包含有元素名称的起始标志，可能还包含有属性设置，一些可选项，和一个结束标志。当然，JSP页面的起始标记和结束标记一定要在同一个页面中，有的元素也仅仅有一个包含属性设置的标志，举例如下：

```
<mytag attr1="attribute value" ...>  
    body  
</mytag>
```

```
<mytag attr1="attribute value" .../>
```

读者会发现，前面讲述的Tomcat的配置文件server.xml和web.xml中已经使用了这种语法形式。

脚本元素则使用的是如ASP般的语法：

```
<%.....%>
```

实际上，每一个JSP页面都应该可以转换为等价的XML页面，在下一章中将详细介绍作为XML的JSP。

JSP元素的属性也和XML中定义的属性遵从同样的原则，JSP页面的属性值一定要使用引号括起来，双引号（"）和单引号（'）都可以使用。另外，作为替代，也可以使用 '

和"来表示双引号和单引号。

4.1.2 JSP中的相对路径

在JSP中，可以使用相对路径来代替绝对路径，在 JSP的语法规范中称之为“URI”，感兴趣的读者可以在RFC2396中找到比较完整的描述，这里举几个例子来说明相对路径的概念：

```
"jspRelativeTest.jsp"  
"/ jspRelativeTest.jsp"  
"/../ jspRelativeTest.jsp"
```

在这三行代码中，都假设是在 c:\tomcat\Jakarta-tomcat\webapps\examples\jsp\test.jsp中使用上述相对路径。

对于第一行代码来说，显然文件 jspRelativeTest.jsp的位置应当为：c:\tomcat\Jakarta-tomcat\webapps\examples\jsp\jspRelativeTest.jsp，这是比较容易理解的。但是对于第二行代码就要注意了，在JSP中，当相对路径以“/”开头的时候，不是相对于网站的根目录，而是相对于包含这个JSP文件的Web应用程序的根目录，也就应当是：c:\tomcat\Jakarta-tomcat\webapps\examples\jspRelativeTest.jsp。如果读者对Web应用程序的概念还不清楚，请参见前几章中的相关章节。第三行代码对于熟悉 UNIX或DOS命令行方式的读者应该不陌生，这种情况下，文件 jspRelativeTest.jsp的位置应当为：c:\tomcat\Jakarta-tomcat\webapps\examples\jsp\jspRelativeTest.jsp。

4.2 注释

一般来说，可以认为在JSP页面中包含有两种不同类型的注释：一种是JSP本身的，用于描述JSP程序代码，另一种是JSP生成的页面的，也就是HTML（或WML...）的注释，用于描述JSP页面执行后的结果页面的功能。HTML页面的注释这里就不说了，下面是JSP本身的注释语法的例子：

```
<%-- 这是一个JSP的注释 --%>
```

实际上，由于在JSP的“脚本代码”部分中，也就是后面会提到的“Scriptlet”直接使用“<% %>”包含起来的部分中可以使用Java的语法，所以下面形式的注释也就理所当然的可以使用了：

```
<% /*这是一个Scriptlet中的注释 */ %>  
<% /**这也是一个Scriptlet中的注释，可以用javadoc从生成的Java文件中提取出注释来 */ %>
```

4.3 指令

前面已经讲过指令在JSP中的地位，指令一般来说有如下的形式：

```
<%@ directive {attr="value"} %>
```

指令的这种语法形式尽管简单明了，但并不是符合XML的，第5章将讲述指令的XML语法。

4.3.1 “page”指令

page指令描述了和页面相关的指示信息。在一个JSP页面中，page指令可以出现多次，但是

每一种属性却只能出现一次，重复的属性设置将覆盖掉先前的设置。

page指令的基本语法格式如下：

```
<%@ page page_directive_attr_list %>
page_directive_attr_list ::= {language=" scriptingLanguage"
                             { extends=" className"
                             { import=" importList"
                             { session="true|false"
                             { buffer="none| sizekb"
                             { autoFlush="true| false"
                             { isThreadSafe="true|false"
                             { info=" info_text"
                             { errorPage=" error_url"
                             { isErrorPage="true|false"
                             { contentType="ctinfo" }
```

下表4-1是对这些属性的解释：

表 4-1

属性名和可选值	说 明
language = " java "	language 变量告诉 server 在文件中将采用哪种语言，在 JSP 当前的规范（JSP1.1）中，java 是 JSP 唯一支持的语法
extends = " package.class "	extends 变量定义了由 JSP 页面产生的 servlet 的父类，一般来说，这个属性不会用到，但是当需要实现一些特殊功能时，也是比较方便的
import = " package.*,package.class " 例如： <%@ page import= " java.util.* " %>	import 变量和任何一个 java 程序的第一部分一样。同样，它总是被放在 JSP 文件的顶部。import 变量的值是一系列用逗号分开的列表，表明想要引入的包和类 注意： java.lang.* javax.servlet.* javax.servlet.jsp.* javax.servlet.http.* 已经缺省地被 JSP 引擎引入了
session = " true false "	session 变量的缺省值是 true，表示当前页面中将有一个缺省的名为“session”的对象来表示当前会话。“session”对象的类型是： javax.servlet.http.HttpSession
buffer = " none 8kb sizekb "	决定输出流（out 对象）是否需要缓冲，缺省值是 8kb。与 autoFlush 一起使用
autoFlush = " true false "	确定是否自动刷新输出缓冲，如果设成 true，则当输出缓冲区满的时候，刷新缓冲区而不是抛出一个异常
isThreadSafe = " true false "	缺省值是 true，如果多个客户请求发向 JSP 引擎时，可以一次被处理。JSP 程序员要处理同步时共享的状态，以保证同步时确实是安全的。 如果 isThreadSafe 被设成 false，则采用单线程模式控制客户端访问该页 这没有使你脱离异常分枝，然而，在 server 可能的情况下，通过它的判断，创建该页的多个实例运行以处理多个装入的客户请求。而且，这无法保证同一个客户端的连续请求定位到同一个 JSP 页面的实例上，在多个页面请求之间的共享资源和状态必须同步

(续)

属性名和可选值	说 明
info = " text "	页面信息通过页面的 Servlet.getServletInfo()来获得页面可以被访问的内容的类型
ErrorPage = " pathToErrorpage "	给出一个JSP文件的相对路径，这个JSP文件用于处理没被处理的例外。这个JSP文件要把isErrorPage设成true。 需要注意的是：JSP是通过使用 ServletRequest对象的setAttribute()方法将名为javax.servlet.jsp.jspException的对象存储起来实现的。 另外，当 AutoFlush设为true的时候，在 JspWriter中的数据刷新到 ServletResponse中以后，任何从JSP文件到错误处理文件的未捕获异常将无法被正常发送
isErrorPage = " true false "	标志一个页面为错误处理页面。设置为 true时，在这个JSP页面中的缺省对象 exception将被定义，其值将被设定为呼叫此页面的 JSP页面的错误对象。缺省为 false
ContentType = " text/html; charset = ISO - 8859 -1 "	设置JSP文件和最终文件的 mime类型和字符集的类型。这一项必须在文件的前部，任何一个其他字符在文件中出现之前。缺省为： ContentType = " text/html; charset = ISO - 8859 -1 "

4.3.2 Include指令

include指令的作用是包含另一个文件，其语法相当简单：

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

在这个指令中应该使用前面讲述的 JSP的相对路径表示法。需要说明的是，JSP还有另外一种包含其他文件的方法：

```
<jsp:include page=" " />
```

表4-2比较了两者的异同：

表 4-2

语 法	状 态	对 象	描 述
<%@ include file= " " %>	编译时包含	静态	JSP引擎将对所包含的文件进行语法分析
<jsp:include page= " " />	运行时包含	静态和动态	JSP引擎将不对所包含的文件进行语法分析

4.3.3 taglib指令

taglib指令用于指示这个JSP页面所使用的标签库，标签库的具体用法属于 JSP比较高级的内容，这里就先不讨论了，先讲述一下基础语法：

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

表4-3是对各个属性的解释。

表 4-3

属 性	说 明
uri	描述这个标签库位置的 URI，可以是相对路径或绝对路径
tagPrefix	定义了一个指示使用此标签库的前缀，例如将 tagPrefix 设为 myPrefix 时，可以使用下面的语句来使用此标签库中的 myTag 标签：
<myPrefix:myTag>	下面这些前缀已经保留： jsp:, jsp:, java:, javax:, servlet:, sun:, 和 sunw: 目前，空的 tagPrefix 将被忽视

4.4 内置对象

为开发的方便，JSP 中内置了一些对象，不需要预先声明就可以在脚本代码和表达式中随意使用，前面已经接触到的 session 和 exception 就是两个内置对象，表 4-4 详细讲述 JSP 中的这些内置对象：

表 4-4

对 象	类 型	描 述	作 用 域
request	javax.servlet.HttpServletRequest 的子类	客户端的请求，通常是 HttpServletRequest 的子类，如果客户的请求中有参数，则该对象就有一个参数列表	request（用户请求期）
response	javax.servlet.ServletResponse 的子类	JSP 页面的响应，是 HttpServletResponse 的子类 页面的属性和需要通过标准 API 来访问的相关对象（本质上是构成服务器环境来让 JSP 运行的一些对象），以便 JSP 引擎来编译页面。但是，不同 server 对这些属性和对象的实现方式不同	page（页面执行期）
pageContext	javax.servlet.jsp.PageContext	解决方案是 JSP 引擎编译用 factory 类返回的服务器的 PageContext 类的实现方法。PageContext 类和 request、response 对象以及 page 指令的一些属性（errorpage, session, buffer, autoflush）同时被初始化，同时提供 request 请求的相关的对象	page（页面执行期）
session	javax.servlet.http.HttpSession	HTTP session 是与 request 联合的对象	session（会话期）
application	javax.servlet.ServletContext	servlet 的环境通过调用 getServletConfig（）.getContext（）方法获得	application（整个 Web 应用程序运行期）
out	javax.servlet.jsp.JspWriter	代表输出流的对象	page（页面执行期）
config	javax.servlet.ServletConfig	页面的 ServletConfig 对象	page（页面执行期）
page	java.lang.Object	指向页面自身的方式（在 java 代码中多以 this 替代）	Page（页面执行期）
exception	java.lang.Throwable	没有被 Throwable 捕获的错误。传向了 errorpage 的 URI	page（页面执行期）

4.5 脚本元素

在JSP中，主要的程序部分就是脚本元素，其中包括三个部分：声明（Declaration）、表达式（Expression）和代码（Scriptlet）。从功能上讲，声明用于声明一个或多个变量，表达式将是一个完整的语言表达式，而代码部分将是一些程序片断。

三个脚本元素的基本语法都是以一个“<%”开头，而以一个“%>”结尾的。

声明的例子：

```
<%! this is a declaration %>
```

代码的例子：

```
<% this is a scriptlet %>
```

表达式的例子：

```
<%= this is an expression %>
```

脚本元素也具有相应的XML兼容语法，将在第6章介绍。

4.5.1 声明

JSP中的声明用于声明一个或多个变量和方法，并不输出任何的文本到 out 输出流去。在声明元素中声明的变量和方法将在 JSP 页面初始化时初始化。

语法为：

```
<%! declaration(s) %>
```

举例如下：

```
<%! int i = 0; %>
```

```
<%! public String f(int i) { if (i<3) return("..."); ... } %>
```

实际上，声明变量和方法的语句完全可以放在 Scriptlet 中，两者有什么不一样呢？放在 <%!.....%> 中的声明语句在编译为 Servlet 的时候将作为类的属性而存在，而放在 Scriptlet 中的声明将在类的方法内部被声明。

4.5.2 表达式

JSP 中的表达式可以被看作一种简单的输出形式，需要注意的是，表达式一定要有一个可以输出的值才行。

语法为：

```
<%= expression %>
```

举例如下：

```
<%= (new java.util.Date()).toLocaleString() %>
```

4.5.3 脚本代码

所谓脚本代码，就是 Scriptlet，也就是 JSP 中的代码部分，在这个部分中可以使用几乎任何

Java的语法。

语法为：

```
<% scriptlet %>
```

举例如下：

```
<%
    if (Calendar.getInstance().get(Calendar.AM_PM) == Calendar.AM) {
%>
    Good Morning
<%
    } else {
%>
    Good Afternoon
<%
    }
%>
```

4.6 动作

动作可以影响输出的文本流，使用、编辑、建立对象。在 JSP中，有一些基本的动作，用户也可以添加自己的动作，这需要使用标签库的知识。JSP中的动作是完全基于XML的，下面来看看JSP由哪些标准的动作以及具有哪些属性。

4.6.1 id和scope属性

id属性和scope属性是每一个JSP动作都具有的属性，其中id表示一个动作的名称，而scope则表示一个动作的作用域。

scope作用域的取值如表4-5所示。

表 4-5

作用域取值	有效范围
page	由javax.servlet.jsp.PageContext得到在用户请求此页面的过程中有效
request	由ServletRequest.getAttribute(name)得到在用户的整个请求过程中有效
session	由HttpSession.getValue(name)得到在用户的整个会话期内有效
application	由ServletContext. getAttribute(name)得到在Web应用程序执行期间有效

4.6.2 标准动作

JSP规范书中规定了一些标准的动作，凡是符合 JSP规范的JSP引擎都应当实现这些标准的动作，下面将一一介绍JSP1.1中规定的标准动作。

1. <jsp:useBean>

<jsp:useBean>大概是JSP中最重要的一个动作，使用这个动作，JSP可以动态使用JavaBeans组件来扩充JSP的功能，由于JavaBeans在开发上及<jsp:useBean>在使用上的简单明了，使得JSP

的开发过程和以往其他动态网页开发工具有了本质上的区别。尽管 ASP等动态网页技术也可以使用组件技术，但是由于 ActiveX控件编写上的复杂和使用上的不方便，实际的开发工作中组件技术使用得并不多。

<jsp:useBean>的语法如下：

```
<jsp:useBean id=" name" scope="page|request|session|application"
    typeSpec />
typeSpec ::= class=" className" |
            class=" className" type=" typeName" |
            type=" typeName" class=" className" |
            beanName=" beanName" type=" typeName" |
            type=" typeName" beanName=" beanName" |
            type=" typeName"
```

如果在<jsp:useBean>中需要加入其他的元素，那么使用下面的语法：

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

这里有几个语法的例子：

```
<jsp:useBean id="connection" class="com.myco.myapp.Connection" />
<jsp:useBean id="connection" class="com.myco.myapp.Connection">
    <jsp:setProperty name="connection" property="timeout" value="33">
/jsp:useBean>
```

在下面的这个例子中，这个JavaBeans对象具有会话期作用域，并且在当前会话中已经存在了。

```
<jsp:useBean id="wombat" type="my.WombatType" scope="session"/>
```

如果这个对象不存在的话，将抛出一个 ClassCastException异常。

2. <jsp:setProperty>

<jsp:setProperty>动作用于向一个JavaBean的属性赋值，需要注意的是，在这个动作中将会使用到的name属性的值将是一个前面已经使用 <jsp:useBean>动作引入的JavaBean的名字。

表4-6说明了在使用<jsp:setProperty>时的类型转换，不过在客户端请求时使用<jsp:setProperty>设定JavaBean的属性可以使用任何类型，JSP文件的执行中也不会自动地进行类型转换。

表 4-6

属 性 类 型	由String类型转换所使用的方法
boolean	java.lang.Boolean.valueOf(String)
Boolean	
byte	java.lang.Byte.valueOf(String)
Byte	
int	java.lang.Integer.valueOf(String)
Integer	
char	java.lang.Character.valueOf(String)
Character	

(续)

属性类型	由String类型转换所使用的方法
double	java.lang.Double.valueOf(String)
Double	
float	java.lang.Float.valueOf(String)
Float	
long	java.lang.Long.valueOf(String)
Long	

<jsp:setProperty>的语法如下：

```
<jsp:setProperty name=" beanName" prop_expr />
prop_expr ::= property="*" |
             property=" propertyName" |
             property=" propertyName" param=" parameterName" |
             property=" propertyName" value=" propertyValue"
propertyValue ::= string
```

表4-7是属性及其解释

表 4-7

属性名	描述
property	此属性表明了需要设定值的 JavaBean 属性的名称。 这里有一个很有意思的特殊的 property 设定：当一个 property 设定为 “*” 时，JSP 解释器将把系统 ServletRequest 对象中的参数一个一个的列举出来，检查这个 JavaBean 的属性是否和 ServletRequest 对象中的参数有相同的名称。如果有，就自动将 ServletRequest 对象中参数值传递给相应的 JavaBean 属性
param	这个属性表明了由系统的 Request 向 JavaBean 传递参数时具体采用哪一个 Request。具体到 Web 页面，也就是哪一个 Form
value	这个属性表明了需要设定给 JavaBean 属性的值，可以是直接赋值，也可以是 ServletRequest 对象的一个参数名

下面就 <jsp:setProperty> 动作举几个例子：

将 ServletRequest 对象 request 中的参数全部输入到名为 request 的 JavaBean 中：

```
<jsp:setProperty name="request" property="*" />
```

将 ServletRequest 对象 user 中的参数 username 输入到名为 user 的 JavaBean 中：

```
<jsp:setProperty name="user" property="user" param="username" />
```

将值 “i+1” 计算出来后输入到名为 results 的 JavaBean 的属性 row 中：

```
<jsp:setProperty name="results" property="row" value="<%= i+1 %>" />
```

3. <jsp:getProperty>

<jsp:getProperty> 动作用于从一个 JavaBean 中得到某个属性的值，无论原先这个属性是什么类型的，都将被转换为一个 String 类型的值。

语法如下：

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

例如：

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

4. <jsp:include>

<jsp:include>用于引入一个静态或动态的页面到一个JSP文件中，这动作仅仅和JspWrite对象发生关系。

<jsp:include>动作可以包含一个或几个<jsp:param>子动作用于向要引入的页面传递参数。

语法如下：

```
<jsp:include page=" urlSpec" flush="true"/>
```

或

```
<jsp:include page=" urlSpec" flush="true">
  { <jsp:param .... /> }
</jsp:include>
```

属性flush设定是否自动刷新缓冲区，实际上，在当前的JSP版本（1.1）中，flush设为false是没有任何意义的。

下面是实例：

```
<jsp:include page="/templates/copyright.html"/>
```

5. <jsp:forward>

<jsp:forward>用于引导客户端的请求到另一个页面或者是另一个Servlet去。

<jsp:forward>动作可以包含一个或几个<jsp:param>子动作，用于向要引导进入的页面传递参数。

需要注意，当<jsp:forward>动作发生的时候，如果已经有文本被写入输出流而且页面没有设置缓冲，那么将抛出一个IllegalStateException的异常。

下面是<jsp:forward>的语法：

```
<jsp:forward page=" relativeURLspec" />
```

或

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

举例如下：

```
<%
    String whereTo = "/templates/"+someValue;
%>
<jsp:forward page='<%= whereTo %>' />
```

6. <jsp:param>

<jsp:param>实际上提供了名称与值的一种一一对应关系，在<jsp:include>、<jsp:forward>和<jsp:plugin>中常常作为子动作使用。

语法为：

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

7. <jsp:plugin>

<jsp:plugin>动作为 Web 开发人员提供了一种在 JSP 文件中嵌入客户端运行的 Java 程序（如：Applet、JavaBean）的方法。在 JSP 处理这个动作的时候，将根据客户端浏览器的不同，JSP 在执行以后将分别输出为 OBJECT 或 EMBED 这两个不同的 HTML 之素。

下面是<jsp:plugin>的语法：

```
<jsp:plugin type="bean|applet"
    code=" objectCode"
    codebase=" objectCodebase"
    { align=" alignment" }
    { archive=" archiveList" }
    { height=" height" }
    { hspace=" hspace" }
    { jreversion=" jreversion" }
    { name=" componentName" }
    { vspace=" vspace" }
    { width=" width" }
    { nspluginurl=" url" }
    { iepluginurl=" url" } >
    { <jsp:params>
    { <jsp:param name=" paramName" value=" paramValue" /> }+
    </jsp:params> }
    { <jsp:fallback> arbitrary_text </jsp:fallback> }
</jsp:plugin>
```

表4-8是对<jsp:plugin>动作的子动作和属性的详细说明：

表 4-8

子动作或属性	说 明
<jsp:param>	设定Java Applet或JavaBean执行所需要的参数
<jsp:fallback>	设定当浏览器不支持此项 PlugIn时候应当显示的内容
type	指示这个对象是一个Java Applet还是一个JavaBean
jreversion	指示这个插件对象执行所需要的JRE版本，缺省情况为“1.1”
nspluginurl	指示对于Netscape Navigator的JRE插件的下载地址（URL）
iepluginurl	指示对于Internet Explorer的JRE插件的下载地址（URL）
code、codebase、align、archive、height、hspace、name、vspace、title、width	和HTML中的意义一致