# JSP 指令

JSP 指令（directive）影响由 JSP 页面生成的 servlet 的整体结构。下面的模板给出指令的两种可能形式。属性值两边的双引号可以替换为单引号，但引号标记不能完全省略。如果要在属性值中使用引号，则要在它们之前添加反斜杠，' 使用 \'，" 使用 \"

**<% directive attribute="value" %>**

**<% directive attribute1="value1"**

**attribute2="value2"**

**......**

**attribute3="value3" %>**

在 JSP 中，主要有3种类型的指令：page，include 和 taglib。page 指令允许通过类的导入、servlet 超类的定制、内容类型的设置、以及诸如此类的事物来控制 servlet 的结构。page 指令可以放在文档中的任何地方。第二个指令，include 允许在 JSP 文件转换到 servlet 时，将一个文件插入到 JSP 页面中。include 指令应该放 置在文档中希望插入文件的地方。第三个指令，taglib，定义自 定义的标记标签。

**一、JSP page 指令**

page 指令可以定义下面这些大小写敏感的属性（大致按照使用的频率列出）：import、contentType、pageEncoding、session、isELIgnored（只限 JSP 2.0）、buffer、autoFlush、info、errorPage、isErrorPage、isThreadSafe、language 和 extends。

2.1 import 属性

使用 page 指令的 import 属性指定 JSP 页面转换成的 servlet 应该输入的包。在 JSP 中，包是绝对必需的。原因是，如果没有使用包，系统则认为所引用的类与当前类在同一个包中。例如，假定一个 JSP 页面包含下面的 scriptlet：

**<% Test t=new Test(); %>**

在此，如果 Test 在某个输入包中，则没有歧义。但是，如果 Test 不在包中，或者页面没有明确地导入 Test 所属的包，那么系统将会认为 Test 就在这个自动生成的 servlet 所在的包中。但问题是自动生成的 servlet 所在的包是未知的！服务器在创建 servlet 时，常常会根据 JSP 页面所在的目录来决定它的包。别的服务器可能使用其他不同的方式。因此，不能指望不使用包的类能够正常工作。对于 bean 也同样如此，因为 bean 不过是遵循某些简单命名约定和结构约定的类。

默认情况下，servlet 导入 java.lang.\*、javax.servlet.\*、javax.servlet.jsp.\*、javax.servlet.http.\*，也许还包括一些服务器特有的包。编写 JSP 代码时，绝不要依靠任何自动导入的服务器特有类。这样做会使得代码不可移植。

使用 import 属性时，可以采用下面两种形式：

**<%@ page import="package.class" %>**

**<%@ page import="package.class1, ..., package.classN" %>**

例如，下面的指令表示 java.util 包和 cn.foololdfat 包中的所有类在使用时无需给出明确的包标识符：

**<%@ page import="java.uti1.\*, cn.foololdfat.\*" %>**

import 是 page 的属性中惟一允许在同一文档中多次出现的属性。尽管 page 指令可以出现在文档中的任何地方，但一般不是将 import 语句放在文档顶部附近，就是放在相应的包首次使用之前。

2.2 contentType 和 pageEncoding 属性

contentType 属性设置 Content-Type 响应报头，标明即将发送到客户程序的文档的 MIME 类型。有关MIME 类型的更多信息，参见Java Web 开发 (5) HTTP 响应报头。

使用 contentType 属性时，可以采用下面两种形式：

**<%@ page contentType="MIME-TYPE" %>**

**<%@ page contentType="MIME-Type; charset=Character-Set" %>**

例如，指令

**<%@ page contentType="application/vnd.ms-excel" %>**

和下面的 scriptlet 所起到的作用基木相同

**<% responce.setContentType("application/vnd.ms-excel"); %>**

两种形式的第一点不同是，response.setContentType 使用明确的 Java 代码（这是一些开发人员力图避免使用的方式），而 page 指令只用到 JSP 语法。第二点不同是，指令被特殊处理，它们不是在出现的位置直接成为 \_jspService 代码。这意味着 response.setContentType 能够有条件地调用，而 page 指令不能。条件性地设置内容的类型主要用在同一内容能够以多种不同的形式进行显示的情况下。

不同于常规 servlet （默认的 MIME 类型为 text/plain），JSP 页面的默认 MIME 类型是 text/html （默认字符集为 ISO-8859-1）。因此，如果 JSP 页曲以 Latin 字符集输出 HTML则根本无需使用 contentType，如果希望同时更改内容的类型和字符集，可以使用下面的语句：

**<%@ page contentType="someMimeType; charset=someCharacterSet" %>**

但是，如果只想更改字符集，使用 pageEncoding 属性更为简单。例如，中文 JSP 页面可以使用下面的语句：

**<%@ page pageEncoding="GBK" %>**

可考虑将Java Web 开发 (5) HTTP 响应报头中生成 Excel 表格的例子用 JSP 实现，以体会 contentType 的作用。

2.3 session 属性

session 属性控制页面是否参与 HTTP 会话。使用这个属性时，可以采用下面两种形式：

**<%@ page session="true" %> <%--Default--%>**

**<%@ page session="false" %>**

true 值（默认）表示，如果存在已有会话，则预定义变量 session （类型为HttpSession）应该绑定到现有的会话；否则，创建新的会话并将其绑定到 session。false 值表示不自动创建会话，在 JSP 页面转换成 servlet 时，对变量 session 的访问会导致错误。

对于高流量的网站，使用 session="false" 可以节省大量的服务器内存。但要注意，session="false" 并不禁用会话跟踪，它只是阻止 JSP 页面为那些尚不拥有会话的用户创建新的会话。由于会话是针对用户，不是针对贞面，所以，关闭某个页面的会话跟踪没有任何益处，除非有可能在同一客户会话中访问到的相关页面都关闭会话跟踪。

2.4 isELIgnored 属性

isELIgnored 属性控制的是：忽略（true）JSP 2.0 表达式语言（EL），还是进行正常的求值（false）。这是 JSP 2.0 新引入的属性；在只支持 JSP 1.2 及早期版本的服务器中，使用这项属性是不合法的。这个属性的默认值依赖于 Web 应用所使用的 web.xml 的版本。如果 web.xml 指定 servlet 2.3（对应JSP 1.2）或更早版本，默认值为 true（但变更默认值依旧是合法的，JSP 2.0 兼容的服务器中都允许使用这项属性，不管 web.xml 的版本如何）。如果 web.xml 指定 servlet 2.4（对应 JSP 2.0）或之后的版本，那么默认值为false。使用这个属性时，可以采用下面两种形式：

**<%@ page isELIgnored="false" %>**

**<%@ page isELIgnored="true" %>**

2.5 buffer 和 autoFlush 属性

buffer 属性指定 out 变量(类型为 JspWriter )使用的缓冲区的大小。使用这个属性时，可以采用下面两种形式：

**<%@ page buffer="sizekb" %>**

**<%@ page buffer="none" %>**

服务器实际使用的缓冲区可能比指定的更大，但不会小于指定的大小。例如，<%@ page buffer="32kb" %> 表示应该对文档的内容进行缓存，除非累积至少为32KB、页面完成或明确地对输出执行清空（例如使用 response. flushBuffer），否则不将文档发送给客户。

默认的缓冲区大小与服务器相关，但至少8KB。如果要将缓冲功能关闭，应该十分小心：这样做要求设置报头或状态代码的 JSP 元素都要出现在文件的顶部，位于任何 HTML 内容之前。另一方面，有时输出内容的每一行都需要较长的生成时间，此时禁用缓冲或使用小缓冲区会更有效率；这样，用户能够在每一行生成之后立即看到它们，而不是等待更长的时间看到成组的行。

autoFlush 属性控制当缓冲区充满之后，是应该自动清空输出缓冲区（默认），还是在缓冲区溢出后抛出一个异常（autoFlush="false"）。使用这个属性时，可以采用下面两种形式：

**<%@ page autoFlush="true" %> <%--Default--%>**

**<%@ page autoFlush="false" %>**

在 buffer="none" 时，false 值是不合法的。如果客户程序是常规的 Web 浏览器，那么 autoFlush="false" 的使用极为罕见。但是，如果客户程序是定制应用程序，可能希望确保应用程序要么接收到完整的消息，要么根本没有消息。false 值还可以用来捕获产生过多数据的数据库查询，但是，一般说来，将这些逻辑放在数据访问代码中（而非表示代码）要更好一些。

2.6 info 属性

info 属性定义一个可以在 servlet 中通过 getServletInfo 方法获取的字符串，使用 info 属性时，采用下面的形式：

**<%@ page info="Some Message" %>**

2.7 errorPage 和 isErrorPage 属性

errorPage 属性用来指定一个 JSP 页面，由该页面来处理当前页面中抛出但未被捕获的任何异常（即类型为 Throwable 的对象）。它的应用方式如下：

**<%@ page errorPaqe="Relative URL" %>**

指定的错误页面可以通过 exception 变量访问抛出的异常。

isErrorPage 属性表示当前页是否可以作为其他 JSP 页面的错误页面。使用 isErrorPage 属性时，可以采用下面两种形式：

**<%@ page isErrorPage="true" %>**

**<%@ page isErrorPage="false" %> <%--Default--%>**

2.8 isThreadSafe 属性

isThreadSafe 属性控制由 JSP 页面生成的 servlet 是允许并行访问（默认），还是同一时间不允许多个请求访问单个 servlet 实例（isThreadSafe="false"）。使用 isThreadSafe 属性时，可以采用下面两种形式：

**<%@ page isThreadSafe="true" %> <%--Default--%>**

**<%@page isThreadSafe="false" %>**

遗憾的是，阻止并发访问的标准机制是实现 SingleThreadModel接口。尽管在早期推荐使用 SingleThreadModel 和 isThreadSafe="false"，但最近的经验表明 SingleThreadModel 的设计很差，使得它基本土毫无用处。因而，应该避免使用 isThreadSafe，采用显式的同步措施取而代之。

2.9 extends 属性

extends 属性指定 JSP 页面所生成的 servlet 的超类（superclass ）。它采用下面的形式：

**<%@ page extends="package.class" %>**

这个属性一般为开发人员或提供商保留，由他们对页面的运作方式做出根本性的改变（如添加个性化特性）。一般人应该避免使用这个属性，除非引用由服务器提供商专为这种目的提供的类。

2.10 language 属性

从某种角度讲，language 属性的作用是指定页面使用的脚本语言，如下所示：

**<%@ page language="java" %>**

就现在来说，由于 Java 既是默认选择，也是惟一合法的选择，所以没必要再去关心这个属性。

二、include指令：在页面转换期间将文件包含到jsp页面。

include指令用于在jsp页面中静态包含一个文件，该文件可以是jsp页面，html页面，文本文件或一段Java代码。

在主页面“转换为servlet前”将jsp代码插入。被包含的页面发生更改的时候，主页面也要更新（转译）。

对于文件包含，使用jsp:include。仅在所包含的文件中定义了主页面需要的字段和方法，或者所包含的文件设置了主页面的响应报头时，才使用include指令。

1，include语法：

**<%@ include file="url"%>**

file的属性值被解释为相对于当前jsp文件的URL.

2，在被包含的文件中不要使用和主页重复的HTML标签。这样会影响原jsp文件中同样的标签导致错误。

因为原文件和被包含文件可以互相访问彼此定义的变量和方法，所以要避免变量和方法在命名上的问题。

3，在jsp中，只有3种功能可以将外部内容包含到JSP文档中。

a，jsp:include动作//该标识被执行时，程序会将请求转发（不是请求重定向）到被包含

【两个页面】页面，并将执行结果输出到浏览器中，然后返回包含页继续执行后面的代码。

b，include指令//最终编译-——>sevlet(一个页面)

c，jsp:plugin动作

4，include指令：

服务器将被包含文件内容每字节插入到主页面中，然后将产生的页面作为单个JSP页面进行处理。

5，jsp:include动作：在“请求”期间将其他页面的“输出”包含进来。这样的行为和RequestDispatcher类的include方法完全相同。

优点：包含页面被更改，无需对主页面做修改。

缺点：被包含页面不能使用任何有可能影响主页面的jsp构造。

也就是不能将完整的HTML文档作为被包含页面，只能含有适合于出现在文件插入点的HTML标签。

参数(使用jsp:param动作元素增加请求参数)：

在被包含页面与主页面使用相同的请求对象。可以在被包含页面“看到”主页面的参数，而在主页面“看不到”被包含页面的参数。

a，page属性：指定所包含的页面，应该是指向某种资源的相对URL.

相对URL不以/开头，则表示为相对于主页面的位置。以/开头表示相对于WEB应用的根目录。

如果由服务器来处理该资源，按相对于当前web应用进行解释；只有浏览器处理该资源时，按相对于服务器的根目录来解释。

对于/开头的URL，服务器总是相对于当前web应用对它进行解释，浏览器总是相对于服务器的根目录来对它进行解释。

可以将要包含的页面放在WEB-INF目录下面，对客户端隐藏。

b，flush属性：指定将页面包含进来之前是否应该清空主页面的输入流(默认false)。

**6，include指令和jsp:include动作的区别：**

调用的时间：

jsp:include动作在“请求期间”被包含。

include指令在“转换期间”被包含。

包含的内容：

jsp:include动作是包含jsp页面或servlet的“输出”和HTML文本。

include指令是文件的实际内容。

最终转换的servle个数：

jsp:include动作在请求期间，会将被请求页面转译为独立的servlet。和主页面的servlet对应。

include指令在转换期间将被包含页面的内容插入到主页面中，得到主页面转译成的servlet。

可否设置影响主页面的响应报头和参数：

jsp:incurred动作不可以，include指令可以。

更新：

jsp:include动作是在请求期间被插入所以他的更改不需要更新主页面。

include指令的更新意味着主页面servlet的更新。

**三、taglib指令：声明用户使用的自定义标签：**

声明用户使用自定义的标签，将标签库描述符文件导入到jsp页面。

taglib指令语法：

**<%@ taglib (uri="tigLibURL" 或 tagDir="tagDir") prefix="tagPrefix" %>**

uri属性:定位标签库描述符的位置。唯一标识和前缀相关的标签库描述符，可以使用绝对或相对URL。

tagDir属性：指示前缀将被用于标识在WEV-INF/tags目录下的标签文件。

prefix属性：标签的前缀，区分多个自定义标签。不可以使用保留前缀和空前缀，遵循XML命名空间的命名约定。

# JSP 指令2

注：一、指令元素(directive elements)：控制所生成的servlet结构。

JSP指令用于“转换阶段”提供整个JSP页面的相关信息，影响由JSP页面生成的Servlet的整体结构。

指令不会产生任何的输出到当前的输出流中。

1，指令元素语法：

<%@ directive {attr="value"}\* %>

说明：

<和% 以及和@，%和>之间不能有任何的空格。

属性值两边的双引号可以替换为单引号。引号标记不能完全省略。

在属性中使用引号使用\进行转义。

2，指令的xml语法：

<jsp:directive.directiveType attr="value1" attr="value2" ... />

在编写XML兼容的jsp页面时，只要不在同一页面中混用xml语法和标准语法，就可以使用指令的xml兼容模式。

3，指令的三种类型：

page，include，taglib。

二、page指令 : 用于设置JSP页面的属性，这些属性将用于和JSP容器通信，控制所生成的servlet结构。

page指令作用整个JSP页面，可以将一个指令放在文档中任何地方。

a，page指令的语法：

<%@ page attr="value1" attr="value2" ... %>

xml语法格式的page指令：

<jsp:directive.page attr="value1" attr="value2" ... />

b，page指令的13个属性

1，import属性：page指令中唯一容许在同一文档出现多次的属性。属性的值可以以逗号隔开。

指定jsp页面转换成servlet应该输入的包。对于没有明确指定包的类，将根据jsp页面所在的包(生成的servlet的目录)决定类的包的位置。

这意味着，将在jsp页面所在的包来寻找该类。

一定要将实用工具类和bean包放在指定的包中。不要依赖于自动导入。

2，language属性：

用于指定在脚本元素中使用的脚本语言，默认java。在jsp2.0规范中，只能是java。

3，contentType和pageEncoding属性：

contentType属性设置发送到客户端文档的响应报头的MIME类型和字符编码。多个使用;号分开。

pageEncodeing属性只用于更改字符编码。

servlet默认MIME是text/plain，jsp默认MIME是text/html。

3，session属性：控制页面是否参与会话

默认true。如果存在已有会话，则预定义session变量，绑定到已有会话中。否则创建新会话将其绑定到session。

对于高流量网站，设置false可以节省大量服务器内存。

设置false表示不自动创建新会话，在jsp页面转换为servlet时，这时对变量session的访问导致错误。

设置为false并不是禁用会话跟踪，它只是阻止jsp页面为不拥有会话的用户创建新会话。

\* 对于不需要会话跟踪的页面那就设置为false；当设置为false时session对象是不可访问的。

4，isELlgnored属性:

定义在jsp页面中是否执行或忽略EL表达式。true表示忽略，false表示执行。

默认值依赖于web.xml的版本。servlet2.3之前默认true，servlet2.4默认false。

用于JSP版本不一致造成使用EL表达式出现的问题。使用:isELlgnored="true";

5，buffer和autoFlush属性：

buffer属性指定out对象(JspWriter)使用的缓冲区大小，以kb为单位，默认8kb。

none表示不使用缓冲区。这样要求设置报头或状态代码的jsp元素要出现在文件的顶部，任何HTML内容之前。

autoFlush属性控制当缓冲区满了后，是自动清空输出缓冲区(默认true)，还是在缓冲区溢出后抛出异常(false)。

在buffer=none的时候，autoFlush=false是错误的。

6，info属性：

定义一个可以在servlet中通过getServletInfo方法获取的字符串。

jsp容器做的是在servlet中生成getServletInfo方法返回info属性指定的String。

7，errorPage和isErrorPage属性：指定页面专用的错误页面。

errorPage属性用来指定一个jsp页面，由该页面来处理当前页面中抛出但没有捕获的任何异常。指定的页面可以通过exception变量访问异常信息。

isErrorPage属性表示当前页是否可以作为其他jsp页面的错误页面。true或false。

错误页面应该放在WEB-INF目录下面，只让服务器访问，也不会生成转发的调用，客户端只能看到最初的请求页面URL,看不到错误页面的URL。

如果为整个web应用程序指定错误页面，或为应用中不同类型的错误指定错误处理页面，使用web.xml中的error-page元素。

如果一个页面通过该属性定义了专有的错误页面，那么在web.XML文件中定义的任何错误页面不会被使用。

只能够在错误处理页面中使用错误对象exception。

8，isThreadSafe属性：采用显式的同步来代替该方法。

控制由jsp页面生成的servlet是否允许并发访问(默认true允许)。

这种阻止并发访问的实现是基于SingleThreadModel接口。所以，避免使用isThreadSafe属性。

9，extends属性：避免使用。这是服务器提供商该做的事情。父类必须是对HttJspPage接口的实现类。而HttJspPage是应用于HTTP协议的请求和响应的JspPage扩展接口。

指定JSP页面生成的servlet的超类。

三，JSP之静态include指令、动态Include指令

（一）使用静态include指令

**<%@ page language="java" contentType="text/html; charset=gb2312"%>**

**<!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=ISO-8859-1">**

**<title>Insert title here</title>**

**</head>**

**<body>**

**<%@include file="scriptlet.jsp" %>//一般放在前面**

**</body>**

**</html>**

（二）使用动态Include指令

**<%@ page language="java" contentType="text/html; charset=ISO-8859-1"**

**pageEncoding="ISO-8859-1"%>**

**<!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=ISO-8859-1">**

**<title>Insert title here</title>**

**</head>**

**<body>**

**<jsp:include page="scriptlet.jsp" />**

**</body>**

**</html>**

scriptlet.jsp页面代码如下：

**<%@ page language="java" contentType="text/html; charset=gb2312"%>**

**<!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=ISO-8859-1">**

**<title>Insert title here</title>**

**</head>**

**<body>**

**<%**

**out.write("来自scriptlet.jsp");**

**%>**

**</body>**

**</html>**

（三）静态导入和动态导入的区别：

（1）静态导入（include指令）通过file属性指定被包含的文件，并且file属性不支持任何表达式；动态导入（include动作）通过page属性指定被包含的文件，且page属性支持JSP表达式；

（2）使用静态导入（include指令）时，被包含的文件内容会原封不动的插入到包含页中，然后JSP编译器再将合成后的文件最终编译成一个Java文件；使用动态导入（include动作）包含文件时，当该标识被执行时，程序会将请求转发（不是请求重定向）到被包含的页面，并将执行结果输出到浏览器中，然后返回包含页继续执行后面的代码。因为服务器执行的是多个文件，所以JSP编译器会分别对这些文件进行编译；

（3）使用include指令包含文件时，由于被包含的文件最终会生成一个文件，所以在被包含、包含文件中不能有重名的变量或方法；而include动作包含文件时，由于每个文件是单独编译的，所以在被包含文件和包含文件中重名的变量和方法是不相冲突的。

（4）静态导入是将被导入页面的代码完全融入，两个页面融合成一个整体Servlet，因此被导入页面甚至不需要是一个完整的页面；而动态导入则在Servlet中使用include方法来引入被导入页面的内容；

1. 静态导入时被导入页面的编译指令会起作用；而动态导入时被导入页面的编译指令则失去作用，只是插入被导入页面的body内容。

四，taglib指令：声明用户使用的自定义标签。

声明用户使用自定义的标签，将标签库描述符文件导入到jsp页面。

taglib指令语法：

<%@ taglib (uri="tigLibURL" 或 tagDir="tagDir") prefix="tagPrefix" %>

uri属性:定位标签库描述符的位置。唯一标识和前缀相关的标签库描述符，可以使用绝对或相对URL。

tagDir属性：指示前缀将被用于标识在WEV-INF/tags目录下的标签文件。

prefix属性：标签的前缀，区分多个自定义标签。不可以使用保留前缀和空前缀，遵循XML命名空间的命名约定。