第 3 章

JSP 语法

本章介绍 JSP 的基本语法，包括指令元素、脚本元素、动作元素、注释。指令元素告诉 JSP 容器如何编译 JSP 文件，脚本元素是 JSP 文件中的 Java 代码，动作元素是用来协助处理客户请求的。

3.1 JSP 文件的组成

JSP 是 HTML 和 Java 脚本混合的文本文件，可以处理用户的 HTTP 请求，并返回动态的页面。JSP 由指令元素、脚本元素、动作元素、注释和 HTML 标签构成。

3.1.1 一个典型的 JSP 文件

一个典型的 JSP 文件如下，文件名为 typical.jsp。文件中包含 page 指令、声明、表达式、小脚本，以及 HTML 标签。

```
<%@ page contentType="text/html; charset=GB18030"%>
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=GB18030" />
<title>一个典型的 JSP 文件</title></head>
<%!
    int count = 10;
    String getDate(){
        return new java.util.Date().toString();
    }
%>
<body>
<h2>当前时间是: <%=getDate()%></h2>
<table width="400" border="1" >
<tr>
    <td width="100">学号</td>
    <td width="300">姓名</td>
</tr>
<%
for(int i=1;i<=count;i++){
    %>
```

```
    <tr><td><%=i%></td><td><%="Name"+i%></td></tr>
    <%
}
%>
</table></body></html>
```

3.1.2 分析 JSP 文件中的元素

typical.jsp 文件的内容简单分析如下：

- <%@ page contentType="text/html; charset=GB18030" %>，page 指令用来说明如何编译 JSP 文件。
- <%! %>声明，用来定义类的成员变量和成员方法。
- <% %>Scriptlet(小脚本)，即 Java 代码。
- <%= %>表达式，计算并输出 Java 表达式的值。
- <html></html>，HTML 文档的标识符。
- <head></head>，文档头部。
- <title></title>，文档标题。
- <body></body>，文档正文。
- <h3></h3>，标题 3。
- <table></table>，表格。
- <tr></tr>，表格的一行。
- <td></td>，表格的一个单元格。

3.1.3 JSP 文件的运行结果

在 Eclipse 中新建动态网站项目(Dynamic Web Project)，项目名为 ch03。在项目的 src/main/webapp 目录中新建 JSP 文件，文件名为 typical.jsp。在 Eclipse 中启动服务器 Tomcat，然后打开浏览器并在地址栏输入下面 URL，就可以看到 JSP 文件的运行结果，如图 3-1 所示。

```
http://localhost:8080/ch03/typical.jsp
```

上面给出的是浏览器把 HTML 解释后展现给用户的效果，实际的 HTML 文件的内容如下：

```
<html><head>
<meta http-equiv="Content-Type" content="text/html; charset=GB18030" />
<title>第一个 JSP 文件</title></head>
<body>
<h3>当前时间是: Web Aug 04 16:40:58 CST 2021</h3>
<table width="400" border="1" >
  <tr>
    <td width="100">学号</td>
```

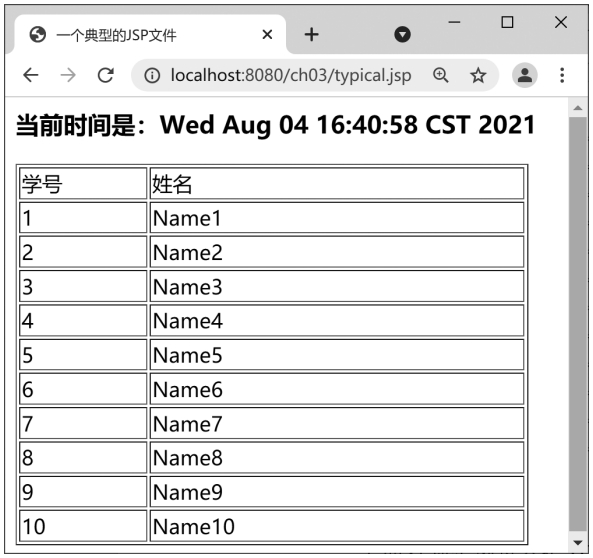

图 3-1　JSP 文件的运行结果

```
    <td width="300">姓名</td>
  </tr>
  <tr><td>1</td><td>Name1</td></tr>
  <tr><td>2</td><td>Name2</td></tr>
  <tr><td>3</td><td>Name3</td></tr>
  <tr><td>4</td><td>Name4</td></tr>
  <tr><td>5</td><td>Name5</td></tr>
  <tr><td>6</td><td>Name6</td></tr>
  <tr><td>7</td><td>Name7</td></tr>
  <tr><td>8</td><td>Name8</td></tr>
  <tr><td>9</td><td>Name9</td></tr>
  <tr><td>10</td><td>Name10</td></tr>
</table></body></html>
```

3.1.4　JSP 转译的 Java 源文件

当服务器上的一个 JSP 页面第一次被请求执行时，服务器上的 JSP 引擎首先将 JSP 页面文件转译成一个 Java 文件，再将 Java 文件编译生成字节码文件，然后通过访问字节码文件实例化的 Java 对象响应客户端的请求。当多个客户请求同一个 JSP 页面时，JSP 引擎为每个客户分配一个线程池里已有的线程，该线程负责访问已经驻留在内存中的 Java 对象来响应客户的请求。

注意：多个线程并发访问同一个 Java 对象，会存在线程安全问题。

typical.jsp 被 Tomcat 转译成 Java 源文件 typical_jsp.java。由于该源文件是 Tomcat 自动转译的，格式比较不规范，下面的源文件经过作者手工排版，并加入了注释。

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import java.io.*;
import org.apache.jasper.runtime.*;
public final class typical_jsp extends HttpJspBase
                                    implements JspSourceDependent {
  //变量声明转译成类的成员变量
  int count = 10;
  //方法声明转译成类的方法
  String getDate(){
    return new java.util.Date().toString();
  }
  private static java.util.List _jspx_dependants;
  public Object getDependants() {
    return _jspx_dependants;
  }
  public void _jspService(HttpServletRequest request,
                      HttpServletResponse response)
                      throws IOException, ServletException {
    JspFactory _jspxFactory = null;
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
    try {
      _jspxFactory = JspFactory.getDefaultFactory();
      response.setContentType("text/html; charset=GB18030");
      pageContext = _jspxFactory.getPageContext(this, request, response,
            null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;
      out.write("\r\n");
      out.write("<html>\r\n");
      out.write("<head>\r\n");
```

```
        out.write("<meta http-equiv=\"Content-Type\" ");
        out.write("content=\"text/html; charset=GB18030\" />\r\n");
        out.write("<title>一个典型的 JSP 文件</title>\r\n");
        out.write("</head>\r\n");
        out.write("<body>\r\n");
        out.write("<h2>当前时间是: ");
        out.print(getDate());
        out.write("</h2>\r\n");
        out.write("<table width=\"400\" border=\"1\" >\r\n");
        out.write("<tr>\r\n");
        out.write("  <td width=\"100\">学号</td>\r\n");
        out.write("  <td width=\"300\">姓名</td>\r\n");
        out.write("</tr>\r\n");
        //for 循环
        for(int i=1;i<=count;i++){
          out.write("<tr><td>");
          out.print(i);                //表达式
          out.write("</td><td>");
          out.print("Name"+i);         //表达式
          out.write("</td></tr>\r\n");
        }
        out.write("</table>\r\n");
        out.write("</body>\r\n");
        out.write("</html>\r\n");
      } catch (Throwable t) {
        if(!(t instanceof SkipPageException)){
          out = _jspx_out;
          if (out != null && out.getBufferSize() != 0)
            out.clearBuffer();
          if (_jspx_page_context != null)
             _jspx_page_context.handlePageException(t);
        }
      }finally{
          if (_jspxFactory != null)
             _jspxFactory.releasePageContext(_jspx_page_context);
      }
    }
  }
```

3.2　JSP 中的注释

注释可以增强 JSP 页面的可读性,并易于 JSP 页面的维护。JSP 页面中的注释可分为以下 3 种:

（1）HTML 注释：在标记符号“<%--”和“--%>”之间加入注释内容。例如：

```
<!--    注释内容    -->
```

JSP 引擎会计算注释中的 JSP 表达式，并把 HTML 注释交给用户，因此用户通过浏览器查看 JSP 页面的源文件时，能够看到 HTML 注释。

（2）JSP 注释：在标记符号“<%--”和“--%>”之间加入注释内容。例如：

```
<%--  注释内容 --%>
```

JSP 引擎忽略 JSP 注释，即在编译 JSP 页面时忽略 JSP 注释，因此浏览器用户是无法通过查看源文件看到 JSP 注释的。

（3）Java 注释：在“/*”和“*/”之间可以加入单行或多行注释，在“//”后可以加入单行注释。例如：

```
/*  注释行 1
    注释行 2
    注释行 3
*/
//单行注释内容
```

1. comment.jsp：演示 JSP 中的注释

```
<%@ page contentType="text/html; charset=GB18030" %>
<html><head><title>comment</title></head><body>
<h2>HTML 注释举例</h2>
<!-- HTML 注释  -->
<!-- 含表达式的 HTML 注释 <%= new java.util.Date()%> -->
<h2>JSP 注释举例</h2>
<%-- JSP 注释  --%>
<h2>Java 注释举例</h2>
<%
  /*
     多行注释
  */
  //单行注释
  String developer = "佟强";
%>
</body>
</html>
```

2. comment.jsp：输出的 HTML 内容

```
<html><head><title>comment</title></head><body>
<h2>HTML 注释举例</h2>
<!-- HTML 注释  -->
<!-- 含表达式的 HTML 注释 Web Aug 04 16:50:58 CST 2021 -->
```

```
<h2>JSP注释举例</h2>
<h2>Java注释举例</h2>
</body>
</html>
```

3.3 指令元素

JSP指令元素用来告诉JSP容器如何编译JSP文件,JSP指令元素有三个:页面指令page、包含指令include、标签库指令taglib。

3.3.1 page指令

page指令用来定义JSP文件中的全局属性。一个JSP页面可以包含多个page指令,除了import属性外,其他属性只能出现一次。page指令的作用对整个JSP页面有效,与其书写的位置无关,但习惯把page指令写在JSP页面的最前面。

```
<%@ page
    [language="java"]
    [import="{package.clazz|package.*},..."]
    [contentType="TYPE;charset=CHARSET"]
    [session="true|false"]
    [buffer="none|8kb|sizekb"]
    [autoFlash="true|false"]
    [isThreadSafe="true|false"]
    [info="text"]
    [errorPage="relativeURL]
    [isErrorPage="true|false"]
    [extends="package.class"]
    [isELIgnored="true|false"]
    [pageEncoding="CHARSET"]
%>
```

例如:

```
<%@ page language="java" contentType="text/html; charset=GB18030"
    pageEncoding="GB18030" session="true"
    import="java.io.*,java.sql.*,javax.sql.*,java.util.*" %>
```

在对浏览器的响应中,应用服务器负责通知浏览器使用什么样的方法来处理接收到的信息,这就要求JSP页面必须设置响应的MIME类型和字符集,即设置contentType属性。MIME是Multipurpose Internet Mail Extensions的缩写,是一个互联网标准,扩展了电子邮件标准。在万维网中使用的HTTP协议也使用了MIME的框架,标准被扩展为互联网媒体类型。JSP页面输出HTML的MIME类型是text/html,而其他类型通常使用Servlet输出。如果用户的浏览器不支持某种MIME类型,那么用户的浏览器就

无法用相应的应用程序处理接收到的信息。除 text/html，常见的 MIME 类型还有：

- image/jpeg，JPEG 图像。
- image/png，PNG 图像。
- image/gif，GIF 图形。
- text/plain，普通文本。
- application/zip，ZIP 压缩文件。
- application/pdf，PDF 文件。
- application/octet-stream，任意的二进制数据。
- application/vnd.ms-excel，.xls 格式的 Excel 工作簿。

.xlsx 格式的 Excel 工作簿文件的 MIME 类型是：

```
application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
```

page 指令的属性描述、默认值和示例，如表 3-1 所示。

表 3-1　page 指令的属性描述、默认值和示例

属　性	描　述	默认值	示　例
language	定义要使用的脚本语言，目前只能是"java"	"java"	language＝"java"
import	和一般的 Java import 意义一样，用于引入要使用的类，用逗号","隔开包和类的列表	默认省略	import＝"java.io.*，java.util.HashMap"
session	指定所在页面是否参与 HTTP 会话	"true"	session＝"true"
buffer	指定客户端输出流的缓冲模式。如果为 none，则不缓冲；可指定具体大小，与 autoFlash 一起使用	默认 8KB	buffer＝"64kb"
autoFlush	为 true 则缓冲区满时，输出缓冲区被刷新；为 false 则缓冲区满时，出现运行异常，表示缓冲区溢出	"true"	autoFlush＝"true"
info	关于 JSP 页面的信息，定义一个字符串，可以使用 servlet.getServletInfo()获得	默认省略	info＝"测试页面"
isErrorPage	表明当前页是否为其他页的 errorPage 目标。如果被设置为 true，则可以使用 exception 对象。相反，如果被设置为 false，则不可以使用 exception 对象	"false"	isErrorPage＝"true"
errorPage	定义此页面出现异常时调用的页面	默认省略	errorPage＝"err.jsp"
isThreadSafe	用来设置 JSP 文件是否能多线程使用。如果设置为 true，那么一个 JSP 能够同时处理多个用户的请求；如果设置为 false，一个 JSP 只能一次处理一个请求	"true"	isThreadSafe＝"true"
contentType	定义 JSP 页面响应的 MIME 类型和响应内容的字符编码	text/html；charset＝ISO-8859-1	"text/html；charset＝GB18030"

续表

属　性	描　述	默认值	示　例
pageEncoding	JSP 页面的字符编码	ISO-8859-1	pageEncoding＝"GB18030"
isELIgnored	指定 EL(表达式语言)是否被忽略。如果为 true,则容器忽略"${}"表达式的计算	"false"	isELIgnored＝"true"

3.3.2　include 指令

include 指令通知容器在当前 JSP 页面中指定的位置嵌入其他文件。被包含的文件内容可以被 JSP 解析,这种解析发生在编译期间。

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

其中 filename 为要包含的文件名。需要注意的是,一经编译,内容就不可变,如果要改变内容,必须重新编译 JSP 文件。相比运行时包含,编译时包含的 JSP 执行效率更高。而且,Tomcat 会检测文件的改动,并自动重新编译文件。

如果 filename 以"/"开头,那么路径是参照 Web 应用所在的根路径;如果 filename 是文件名,或者是以目录名开头,那么路径是以 JSP 文件所在路径为参照的相对路径。

例如,设计网站时,为了让网站具有统一的风格,可以将页面头部和页面底部做成公共的页面,如图 3-2 所示。然后在其他页面中使用 include 指令将页面头部和页面底部包含进来。需要注意的是,头部和底部的页面应是 HTML 片段,而不是完整的 HTML 文档。

图 3-2　include 指令举例

1. 页面文件 home.jsp

```
<%@ page contentType="text/html; charset=GB18030" %>
<html><head>
<style type="text/css">
/* body 标签的样式 */
body{
    text-align: center;
    margin:0;
    padding:0;
}
/* 标题 1 去掉 margin 和 padding */
h1{
    margin:0;
    padding:0;
}
```

```
/* 段落去掉 margin 和 padding */
p{
    margin:0; padding:0;
}
/* 定义一个放所有页面内容的容器 */
#container{
    width: 800px;
    height: auto;
    margin: 0  auto;
    text-align:left;
}
/* 定义页面头部 */
#header{
    width: 800px;
    height: 60px;
    line-height: 60px;
    background-color: red;
}
/* 定义页面中部 */
#content{
    width: 800px;
    height:680px;
    background-color: green;
}
/* 定义页面底部 */
#footer{
    width: 800px;
    height: 40px;
    line-height: 40px;
    background-color: blue;
}
</style>
<title>使用 include 指令包含页面头部和页面底部</title>
</head>
<body>
<div id="container">
    <div id="header">
        <%@ include file="header.jsp" %>
    </div>
    <div id="content">
        <h1>文档的标题</h1>
        <p>文档内容</p>
    </div>
    <div id="footer">
```

```
        <%@ include file="footer.jsp" %>
    </div>
</div>
</body>
</html>
```

2. 页面头部文件 header.jsp

```
<%@ page contentType="text/html; charset=GB18030" pageEncoding="GB18030"%>
<h1>页面的头部</h1>
```

3. 页面底部文件 footer.jsp

```
<%@ page contentType="text/html; charset=GB18030" pageEncoding="GB18030"%>
<p>页面的底部</p>
```

3.3.3　taglib 指令

taglib 指令允许页面使用自定义标签。用户可以开发标签库，为标签库编写.tld 配置文件，然后在 JSP 页面中使用 taglib 指令引入标签库。在 JSP 2.0 规范中引入了标准标签库 JSTL。如何使用表达式语言 EL 和标准标签库 JSTL 请分别参考第 11 章表达式语言 EL 和第 12 章标准标签库 JSTL，可提前阅读。JSP 中使用 taglib 指令引入标签库的语法如下：

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

uri 用来告诉 JSP 引擎怎么找到标签描述文件和标签库。prefix 定义在 JSP 页面中引用标签使用的前缀，前缀不可以是：jsp、jspx、java、javax、sun、servlet 和 sunw。

例如，引入 JSTL 核心标签库的 taglib 指令是：

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

引入核心标签库之后，就可以使用 JSTL 核心标签库中定义的标签，例如使用迭代标签<c:forEach>输出表格数据行的代码如下：

```
<table width="600" border="1">
<thead>
  <tr><th>学号</th> <th>姓名</th></tr>
</thead>
<tbody>
  <c:forEach var="student" items="${students}">
    <td>${student.id}</td>
    <td>${student.name}</td>
  </c:forEach>
</tbody>
</table>
```

3.4 脚 本 元 素

JSP 脚本元素是 JSP 最烦琐的元素，特别是 Scriptlet，在早期的 JSP 代码中占主导地位。脚本元素通常是 Java 编写的脚本代码，可以声明变量和方法，可以包含任意的 Java 代码和表达式计算。

3.4.1 声明<%!与%>

在 JSP 中，声明（Declaration）是一段 Java 代码，它用来定义在产生的类文件中类的属性和方法。声明后面的变量和方法可以在 JSP 的任意地方使用。可以声明方法，也可以声明变量。声明的格式如下：

```
<%!
    variable declaration
    method declaration(paramType param,...)
%>
```

1. 变量声明

```
<%!
    int m, n=100, k;
    String message="Hello World!";
    Date date;
%>
```

由“<%!”和“%>”之间定义的变量是类的成员变量，这些变量在整个 JSP 页面内都有效，与“<%!”和“%>”标记符在页面中所在的位置无关，但习惯上通常把“<%!”和“%>”标记符写在 JSP 页面的前面。当多个客户端请求同一个 JSP 页面时，JSP 引擎会为每个客户端分配一个线程，这些线程共享类的成员变量，任何一个客户端对成员变量的修改都将影响其他客户端。因此，声明的成员变量存在线程的安全问题，应该尽量避免使用。

在小脚本中使用上面所声明变量的代码如下：

```
<%
    m = 20;
    k = m+n;
    date = new Date();
    out.println(message);
%>
```

2. 方法声明

```
<%!
    /*将两个整数相加，返回相加的结果*/
```

```
    int add(int x, int y) {
        return x+y;
    }
    /* 计算 n 的阶乘 */
    long factorial(long n) {
        long fact = 1;
        for(long i=2; i<=n; i++) {
           fact *=i;
        }
        return fact;
    }
%>
```

在“<%!”和“%>”之间定义的方法是类的成员方法，在整个 JSP 页面内有效。但方法内部定义的变量是局部变量，方法被调用时才分配局部变量的存储空间，调用完毕即释放所占的内存。

在小脚本中使用上面定义的 add()方法：

```
<%
    int a = 100;
    int b = 23;
    int c = add(a,b);
    out.println("a="+a+" "+"b="+b+" "+"c="+c);
%>
```

在表达式中使用 factorial()方法：

```
<h3>10 的阶乘是<%=factorial(10)%></h3>
```

将以上变量声明和方法声明的代码应用到一个 JSP 文件中(declaration.jsp)，运行结果如图 3-3 所示。

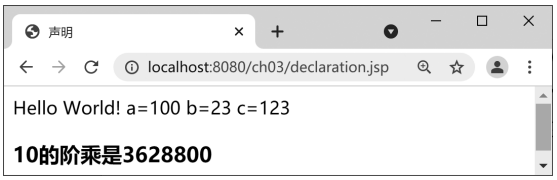

图 3-3　declaration.jsp 的运行结果

3. 页面计数器 page_counter.jsp

变量声明为类的成员变量，多个线程同时访问的是一个对象中的同一个变量。我们利用一个成员变量“counter”实现一个页面计数器，可以验证多次访问对应同一个 Java 对象。

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

如果在页面的小脚本部分直接修改 counter，可能导致线程安全问题。比如第一个线程读取了 counter 的值为 10，还没来得及给 counter 加 1，这时第二个线程读到 counter 的值仍为 10。然后，第一个线程执行加 1 写回操作，counter 的值被更新为 11；第二个线程也执行加 1 写回操作，也把 counter 的值更新为 11。两个线程都执行加 1 操作之后 counter 的取值应该是 12，但是这里却得到了错误结果 11。

为了解决针对一个对象的互斥访问，Java 语言提供了 synchoronized 关键字。Java 虚拟机确保同时只能有一个线程进入有 synchoronized 关键字修饰的方法。定义一个有 synchoronized 关键字修饰的方法来执行 counter 加 1 的操作。

```
<%!
    synchronized int incrementCounter( ) {
        return ++counter;
    }
%>
```

在 JSP 页面中显示页面被访问次数的代码如下：

```
<h2>当前页面被访问计数：<%=incrementCounter()%></h2>
```

将以上代码应用到 page_counter.jsp，每次刷新计数加 1，运行结果如图 3-4 所示。

图 3-4　page_counter.jsp 的运行结果

3.4.2　表达式 <%=与%>

表达式(Expression)是在 JSP 请求处理阶段计算它的值，所得的结果转换为字符串并输出。表达式所在页面的位置，也就是该表达式计算结果显示的位置。表达式的语法是：

```
<%= Java Expression %>
```

表达式必须能求值，且不能以“;”(英文分号)结束，因为表达式不是 Java 的语句。

表达式的示例：expression.jsp

```
<%@ page contentType="text/html; charset=GB18030" %>
<html><head><title>表达式</title></head>
<body>
<%
    //局部变量的定义
    double a=9, b=5, c=16;
    int x=12, y=10;
```

```
%>
<p>ax+y^2-by+c 的值：<%=a * x+y * y-b * y+c%></p>
<p>x&gt;y&amp;&amp;a==b 的值：<%=x>y&&a==b%></p>
<p>sin(x)+cos(y)的值：<%=Math.sin(x)+Math.cos(y)%></p>
<p>16 的平方根：<%=Math.sqrt(16)%></p>
</body>
</html>
```

expression.jsp 的运行结果如图 3-5 所示。

图 3-5　expression.jsp 的运行结果

3.4.3　小脚本 <% 与 %>

小脚本(Scriptlet)是 JSP 页面处理请求时执行的 Java 代码，Scriptlet 包括在“<%”和“%>”之间。它可以产生输出，可以是一些流程控制语句，也可以包含 Java 注释。Scriptlet 的语法是：

```
<%
    Java Statements
%>
```

在“<%”和“%>”之间定义的变量是函数的局部变量。局部变量的有效范围和其定义的位置有关，即局部变量在定义的位置之后才有效。JSP 引擎将 JSP 文件转译成 Java 文件时，将各个小脚本中的局部变量都转化为 Java 类的“_jspService()”方法的局部变量。

1. Java 小脚本(Scriptlet)示例：name.jsp

```
<%@ page contentType="text/html; charset=GB18030" %>
<html>
<head>
<title>小脚本(Scriptlet)</title>
</head>
<body>
<%
   String[] names = {"施耐庵","曹雪芹","佟强","罗贯中"};
   for(int i=0; i<names.length; i++){
       out.print(names[i]);
```

```
        if(names[i].equals("佟强")){
            out.print("是本书的作者。");
        }
        out.println("<br/>");
    }
%>
</body>
</html>
```

name.jsp 的运行结果如图 3-6 所示。

图 3-6　name.jsp 的运行结果

下面再来分析一个复杂一点的小脚本示例，静态和动态都可以在循环时切换。

2. score.jsp

```
<%@ page contentType="text/html; charset=GB18030" import="java.util.*" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=GB18030" />
<title>小脚本(Scriptlet)</title>
</head>
<body>
<% //Map 里面存储的是多组关键字和与其对应的值
    Map<String,Float> students = new HashMap<String,Float>();
    students.put("张三", 78.5F);      //张三,78.5 分
    students.put("李四", 87.0F);      //李四,87 分
    students.put("赵五", 92.5F);      //赵五,92.5 分
%>
<h1 align="center">学生成绩表</h1>
<table width="600" border="1" align="center">
  <thead>
    <tr align="center">
      <th>姓名</th><th>成绩</th>
    </tr>
  </thead>
  <tbody>
  <%
  //获得学生姓名的集合
  Set<String> studentNames = students.keySet();
```

```
    //获得姓名集合的迭代器
    Iterator<String> it = studentNames.iterator();
    //迭代姓名集合
    while(it.hasNext()){
        //取出一个姓名
        String name = it.next();
        //get()方法根据关键字得到对应的值
        Float score = students.get(name);
        %>
          <tr align="center"><td><%=name%></td><td><%=score%></td></tr>
        <%
    }
    %>
    </tbody>
    </table>
</body>
</html>
```

score.jsp 的运行结果如图 3-7 所示。

图 3-7　score.jsp 的运行结果

3.4.4　表达式语言${ }

表达式语言(Expression Language,EL)是 JSP 2.0 增加的技术,通过表达式语言,可以简化 JSP 的开发,使代码整洁。EL 表达式使用“ ${ }”表示,用于读取 pageContext、request、session、application 对象上绑定的属性(Attribute)。

表达式语言可以读取 request 对象上的属性,下面代码在 request 对象上绑定一个属性 message,属性的取值是 String 对象。

```
<%
    request.setAttribute("message","Hello Expression Language!");
%>
```

在 JSP 页面中静态的 HTML 部分使用 EL 读取属性值并输出,可使用 ${message}。

```
<h3>${message}</h3>
```

使用 EL 也可以很方便地读取 JavaBean(绑定在 request 等对象上的自定义 Java 对象)的属性,以下代码创建了一个 Student 对象,并将这个对象绑定到 request 对象上,属性名是“student”。这样,student 就成为了一个 request 范围内的一个 JavaBean,其自身具有属性学号(stuNo)、姓名(name)和成绩(score)。

```
<%
    Student student = new Student("201501001","王小花",96.5F);
    //将 Student 对象绑定到 request 上,属性名是 student
    request.setAttribute("student", student);
%>
```

在 JSP 页面的静态 HTML 部分,可以使用如下 EL 表达式读取 student 对象的属性。但是需要强调的是,EL 表达式并不直接读取 student 对象的数据成员,而是通过调用对应的 Getter 方法来读取数据成员,比如,表达式 ${student.name}将调用 student.getName()方法。

```
<p>${student.stuNo} ${student.name} ${student.score}</p>
```

将以上程序段和 EL 表达式应用到 el.jsp 中,运行结果如图 3-8 所示。另外需要的 Student 类的定义如下:

```
package cn.edu.uibe.domain;
public class Student {
    private String stuNo;              //学号
    private String name;               //姓名
    private float score;               //成绩
    public Student(String stuNo,String name,float score){
        this.stuNo = stuNo;
        this.name = name;
        this.score = score;
    }
    public String getStuNo() { return stuNo; }
    public void setStuNo(String stuNo) { this.stuNo = stuNo; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public float getScore() { return score; }
    public void setScore(float score) { this.score = score; }
}
```

图 3-8　el.jsp 的运行结果

3.5 动作元素

JSP动作元素是在请求处理阶段按照其在页面中出现的顺序执行的，该类元素只有在被执行时才实现其所具有的功能，这与指令元素不同。JSP指令元素是在编译时发生作用，指导JSP引擎将JSP页面转译成Servlet。JSP动作元素使用XML语法，有以下两种格式：

```
<prefix:tag attribute=value attribute-list .../>
```

或者：

```
<prefix:tag attribute=value attribute-list ...>
    ......
</prefix:tag>
```

在JSP页面中，常用的动作元素包括<jsp:include>、<jsp:forward>、<jsp:useBean>、<jsp:setProperty>、<jsp:getProperty>、<jsp:param>。

3.5.1 <jsp:param> 提供参数

<jsp:param>操作是以“名称-值”对的形式为其他标签提供额外参数。它可与<jsp:include>和<jsp:forward>一起使用，在动态包含和转发时为目标JSP页面提供参数。它的使用形式如下：

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

其中name为参数名，value为参数值。例如：

```
<jsp:param name="productId" value="1048485790"/>
```

3.5.2 <jsp:include> 包含页面

<jsp:include>操作在处理用户请求时动态地引入其他资源(JSP、HTML)。被包含的对象只有对JSP Writer对象的访问权，不能设置Header或者Cookie。与include指令相比，<jsp:include>操作发生在处理用户请求时，是动态的；而include指令发生在JSP文件编译时，是静态的。<jsp:include>是运行时调用另外一个JSP页面，并将另外一个页面的运行结果输出到当前页面中include动作元素所在的位置；而<%@ include>是在编译时将被包含的文件复制到include指令所在位置，然后作为一个整体编译。<jsp:include>语法如下：

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

或者：

```
<jsp:include page="url" flush="true">
    <jsp:param name="paramName" value="paramValue"/>
```

```
    ......
</jsp:include>
```

其中，page 属性给出被包含页面的 URL，flush 属性标识当输出缓冲区满时，是否清空缓冲区。page 属性的 url 如果以“/”开头，则从当前应用所在路径开始查找页面文件，如果以文件名或目录名开头，则以当前路径为基础查找被包含的页面文件。flush 属性的默认值是 false，通常情况下设置为 true。<jsp:param>子元素可以向被包含的 JSP 页面传递参数。

以下<jsp:include>的示例由 2 个页面构成，jspinclude1.jsp 页面中使用 include 动作元素动态地包含 jspinclude2.jsp，并使用<jsp:param>传递了参数 message。

1. 动态包含另外一个文件：jspinclude1.jsp

```
<%@ page contentType="text/html; charset=GB18030" %>
<html><head>
<title>jsp:include demo</title>
</head><body>
<h3>包含前</h3>
<jsp:include page="jspinclude2.jsp">
    <jsp:param name="message" value="Hello Include!"/>
</jsp:include>
<h3>包含后</h3>
</body></html>
```

2. 被包含的文件：jspinclude2.jsp

```
<%@ page contentType="text/html; charset=GB18030" %>
<%
    String message = request.getParameter("message");
    out.println("<h3>被包含页面: "+message+"</h3>");
%>
```

3. 分析运行结果

在浏览器中访问第一个 JSP 文件 jspinclude1.jsp，运行结果如图 3-9 所示。

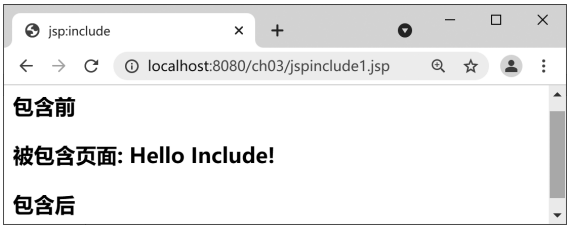

图 3-9　jspinclude1.jsp 的运行结果

需要注意的是，被包含的 JSP 文件只能输出 HTML 片段，而不能是完整的 HTML 文件，否则将导致输出的结果包含重复的<html>、<head>、<title>、<body>等标签。通